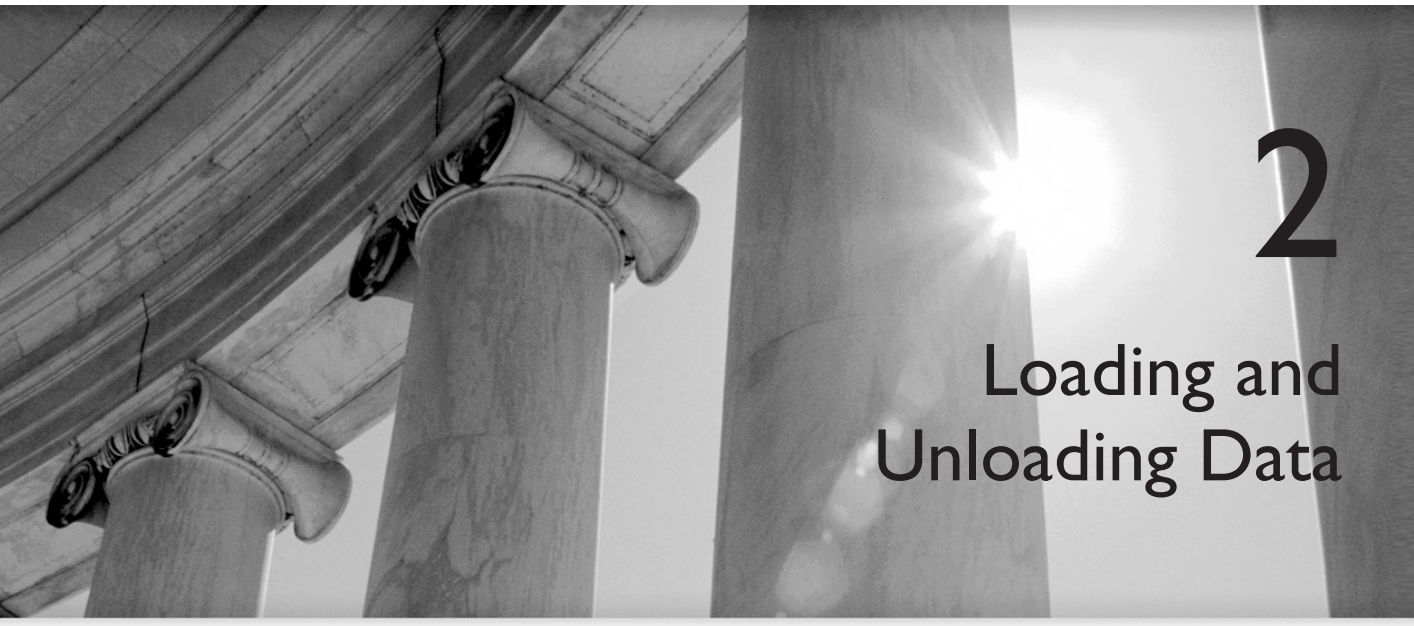


2 Loading and Unloading Data

CERTIFICATION OBJECTIVES

Almost every Oracle DBA is familiar with the traditional Oracle data-loading utilities: export and import. In Oracle Database10*g*, you now have a newer and more refined version of the old export and import utilities, called the Data Pump export and import utilities. The old export and import utilities continue to be available under Oracle Database 10*g*, but Oracle would prefer you to use the Data Pump technology, because it offers you more sophisticated features than the old export/import technology.

While the Data Pump export and import utilities look quite similar to the traditional export and import utilities, the new technology is vastly superior in many ways. For example, you can now interrupt export/import jobs in the middle, and then resume them. You can even restart failed export and import jobs. You can also remap object attributes to modify the objects. You can easily monitor your Data Pump jobs from a different session, and you can even modify job attributes on the fly, during the course of a job. It is easy now to move massive amounts of data quickly, using parallelization techniques. Because Oracle provides you the Application Programming Interfaces (APIs) for the Data Pump technology, you can now easily incorporate export/import jobs within PL/SQL programs.

The Data Pump export and import jobs need more startup time than the old export and import utilities. Therefore, you may still want to use the old export and import utilities for small jobs.

In this chapter, you'll also look at the new features related to transportable tables, as well as enhancements in the external tables feature, which was first introduced in Oracle9*i*. In Oracle Database 10*g*, you can now write to external tables, instead of being merely be able to read from them.

- Introduction to the Data Pump architecture
- Using Data Pump export and import
- Monitoring a Data Pump job
- Creating external tables for data population
- Defining external table properties
- Transporting tablespaces across different platforms

Let's start this very important chapter with a brief introduction to the new Data Pump technology.

CERTIFICATION OBJECTIVE 2.01

Introduction to the Data Pump Architecture

The new Oracle Data Pump facility enables DBAs to transfer large amounts of data and metadata at very high speeds compared to the older export/import technology. Data Pump manages multiple, parallel streams of data to achieve maximum throughput. Oracle claims that Data Pump enables you to decrease total export time by more than two orders of magnitude in most data-intensive export jobs. Imports are supposed to run 15 to 30 times faster than with the original import utility. Both of the above estimates are for single-thread operations; parallel threads will make the operations even faster.

Oracle Data Pump is a complete superset of the original export and import utilities. In addition to all the old capabilities of the export and import utilities, Data Pump also lets you estimate job times, perform fine-grained object selection, monitor jobs effectively, and directly load one database from a remote instance.

For compatibility purposes, Oracle still includes the old export and import utilities in Oracle Database 10*g*. Thus, you can continue to use your export and import scripts as usual, without any changes. Oracle Corporation recommends that you use the Oracle Database10*g* Data Pump export and import, even though the older export and import utilities are still available to you, because of the superior performance provided by the newer tools. Oracle will support the original import utility forever. This means that you'll always have a way of importing dump files from earlier versions of Oracle. However, Oracle will eventually deprecate the original export utility.

The new Data Pump technology lets you export data only to disk. You cannot use a tape drive when performing a Data Pump export.

Oracle Data Pump technology consists of two components: the Data Pump export utility, to unload data objects from a database, and the Data Pump import utility, to load data objects into a database. You access the two Data Pump utilities through a pair of clients called `expdp` and `impdp`. As their names indicate, the first of these corresponds to the traditional export utility and the latter to the import utility. You can control both Data Pump export and import with the help of several parameters. Here's how you invoke the two utilities:

```
$ expdp username/password (various parameters here)
$ impdp username/password (various parameters here)
```

I'm sure you are quite familiar with the interactive mode of using the traditional export and import utilities. In this mode, you enter your choices in response to various prompts. The parameters are the same for the interactive and command-line modes, although you can use only a limited set of export and import parameters during an interactive operation. Unlike in the old export and import utilities, Data Pump utilities have a set of parameters you can use at the command line and a set of special commands you can use only in an interactive mode. I'll explain the main parameters, commands, and the important features of the Data Pump toolset in the following sections. You can also get a quick summary of all Data Pump parameters and commands by simply typing **`expdp help=y`** or **`impdp help=y`** at the command line.

The Data Pump export utility will unload data into operating system files known as *dump files*. It writes to these files in a proprietary format, which only the Data Pump import utility can understand while loading the data in the dump files into the same or another Oracle database. You can take Data Pump export dump files from an operating system and import them into a database running on a different type of platform, as is the case with the older export/import utilities.

The original export and Data Pump dump files aren't compatible. You can't read export dump files with Data Pump and vice versa. The new features of Oracle Database 10g aren't supported in the original export utility, which you'll still have access to in Oracle Database 10g.

In addition to `expdp` and `impdp`, you can have other clients perform Data Pump export and import as well, by using the Data Pump API. The database uses the Oracle-supplied package `DBMS_DATA PUMP` to implement the Data Pump API. Through this package, you can programmatically access the Data Pump export and import utilities. This means that you can create powerful custom data-movement utilities using the Data Pump technology.

The traditional export utility is a normal user process that writes data to its local disks. The old export utility fetches this data from a server process as part of a regular session. By contrast, the Data Pump `expdp` user process launches a server-side process that writes data to disks on the server node, and this process runs independently of the session established by the `expdp` client.

The Data Pump technology is remarkably different from the traditional export and import utilities. In the following sections, you'll learn about those differences as we cover the following topics:

- Benefits of Data Pump technology
- Data Pump components

- Data-access methods
- Data Pump files
- The mechanics of a Data Pump job

Benefits of Data Pump Technology

Data Pump technology offers several benefits over the traditional export and import data utilities. The following are the main benefits of the Data Pump technology:

- **Ability to restart data pump jobs** You can now easily restart jobs that either have stalled due to lack of space or have failed for various reasons. You may also voluntarily stop and restart jobs.
- **Parallel execution capabilities** By simply specifying a value for the `PARALLEL` parameter, you can now choose the number of active execution servers for each export job.
- **Ability to attach to running jobs** You now have the amazing capability to *attach* to a running Data Pump job from a different screen or location. This enables you to monitor jobs, as well as to modify certain parameters interactively. The jobs continue to run while you are attaching to and detaching from them. Data Pump is an integral part of the Oracle Database server, and as such, it doesn't need a client to run once it starts a job.
- **Network mode of operation** Once you create database links between two databases, you can perform exports from a remote database straight to a dump file set. You can also perform direct imports via the network using database links, without using any dump files. The network mode is a means of transferring data from one database directly into another database via SQLNET with the help of database links, without needing to stage it on disk at all.
- **Fine-grained data import capability** Oracle9*i* offered only one parameter that gave you the ability to perform data loads at a fine-grained level. This parameter was `QUERY`, which enabled you to specify that the export utility extract only a specified portion of a table's rows. Now with Data Pump, you have access to a vastly improved fine-grained options arsenal, thanks to new parameters like `INCLUDE` and `EXCLUDE`.

- **Remapping capabilities** During a data pump import, you now have the ability to remap schemas and tablespaces, as well as filenames, by using the new REMAP_ * parameters. Remapping capabilities enable you to modify objects during the process of importing data, by changing old attributes to new values. For example, the REMAP_SCHEMA parameter enables you to map all of user HR's schema to a new user, OE. The REMAP_SCHEMA parameter is similar to the TOUSER parameter in the old import utility.
- **Ability to estimate space requirements** In a Data Pump job, you can now estimate the space requirements of a job by using either the ESTIMATE or the ESTIMATE_ONLY parameter.

Data Pump Components

On the surface, expdp and impdp, the clients for the Data Pump export and Data Pump import utilities, respectively, are quite similar to the exp and imp commands. However, while they are syntactically similar to the Data Pump clients, exp and imp are absolutely ordinary user processes that use SQL SELECT, CREATE, and INSERT commands. In contrast, the new utilities are more like control processes that initiate jobs.

The new Data Pump technology is based entirely on the server; all data movement occurs on the server. The older export and import utilities acted as clients through which all the data movement took place. In Data Pump export and import, the database instance handles the Data Pump utilities.

You can look at the Data Pump technology as consisting of three major components:

- The DBMS_DATA PUMP package
- The DBMS_METADATA package
- The command-line clients, expdp and impdp

The DBMS_DATAPUMP package contains the guts of the Data Pump technology, in the form of procedures that actually drive the data loading and unloading jobs. The contents of this package perform the work of both the Data Pump export and import utilities. In traditional export/import, Oracle uses normal SQL to take the data in the export dump files and inserts it sequentially into the database tables during the import process. In the Data Pump technology, the DBMS_DATA PUMP package performs the export and import of data.

The DBMS_DATA PUMP is the main engine for driving data loading and unloading. To extract and modify data dictionary metadata, Oracle provides the DBMS_METADATA package, which has been available since the Oracle9*i* version. In traditional export

and import utilities, the metadata of the objects is included in the export dump file. In Data Pump technology, you need to use the `DBMS_METADATA` package to extract the appropriate metadata.

Note that both the packages—`DBMS_DATA PUMP` and `DBMS_METADATA`—act as APIs, in the sense that you can use either of them directly in your programs to load and unload data, without accessing the `expdp` and `impdp` clients.

Data-Access Methods

A Data Pump import or export job can access table data in either of two ways, depending on which one is faster for the specific case:

- **Direct path** This access uses the Direct Path API. Direct path exports and imports lead to improved performance, since the direct path internal stream format is the same format as the data stored in Oracle dump files. This leads to a reduced need for data conversions.
- **External tables** The external tables feature lets Oracle read data from and write data to operating system files that lie outside the database.

Since direct-path access doesn't support intra-partition parallelism, external tables are used for very large data loading or unloading jobs.

It is up to Oracle to decide which access method it will employ for a given job. Oracle always tries to first use the direct-path method to load or unload data. Under some conditions, such as the following, it may not able to use the direct method:

- Clustered tables
- Presence of active triggers in the tables
- Export of a single partition in a table with a global index
- Presence of referential integrity constraints
- Presence of domain indexes on LOB columns
- Tables with fine-grained access control enabled in the insert mode
- Tables with BFILE or opaque type columns

In all these cases, the structure of the table and/or the indexes precludes the use of direct-path access, so Data Pump will use external tables. On the other hand, if your table has any LONG data, you must use the direct-path access.

exam Watch

The datafile format is identical in external tables and the direct-access method. Therefore, you can easily export data with one method and import it with the other method, if you wish.

Data Pump Files

As in the case of the traditional export and import utilities, Data Pump uses dump files and other log files, but there are significant differences. You'll use three types of files for Data Pump operations:

- **Dump files** These hold the data for the Data Pump job.
- **Log files** These are the standard files for logging the results of Data Pump operations.
- **SQL files** Data Pump import uses a special parameter called `SQLFILE`, which will write all the Data Definition Language (DDL) statements it will execute during the import job to a file. Data Pump doesn't actually execute the SQL, but merely writes the DDL statements to the file specified by the `SQLFILE` parameter. You use SQL files only to hold the output of the `SQLFILE` command during a Data Pump import job. This parameter is discussed in the "Data Pump Import Parameters" section later in this chapter.

In Data Pump, you use directories and directory objects, unlike in the export and import utilities. The following sections explain how to use directory objects.

Using Directory Objects

Recall that the Data Pump technology is server-based, not client-based. This means that a Data Pump job creates all its dump files on the server, not on the client machine where a job may have originated. Oracle background processes are responsible for all dump file set I/O, on behalf of the privileged user ORACLE. This means that for security reasons, you can't let any user be able to specify an absolute file path on the server. In addition to a possible violation of security, there is the matter of safety, as you can unwittingly overwrite a server file if you are given the power to write dump files anywhere on the system. To avoid these problems, Data Pump uses directory objects.

Directory objects are named objects that Data Pump maps to a specific operating system directory. For example, a directory object named dpump_dir1 can point to

the /u01/app/oracle/admin/export directory on the server. You can then access the export directory by simply using the dpump_dir1 directory object name. Here's how you create a directory object:

```
SQL> CREATE DIRECTORY dpump_dir1 as 'c:/oracle/product/10.1.0/oradata/export';
Directory created.
```

To create a directory, a user must have the DBA role or have the CREATE ANY DIRECTORY privilege.

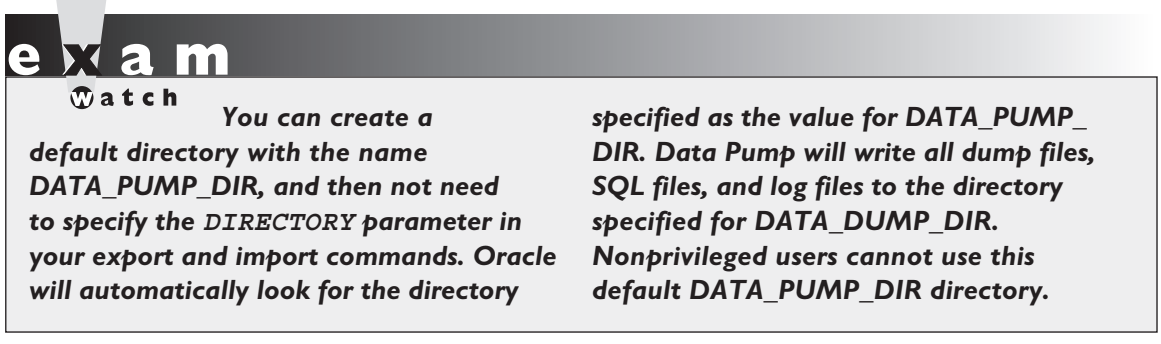
In order for a user to use a specific directory, the user must have access privileges to the directory object. For example, in order to grant user SALAPATI privileges on the new directory dpump_dir1, you need to grant the following privileges:

```
SQL> grant read, write on directory dpump_dir1 to salapati
Grant succeeded.
SQL>
```

You'll need the write privilege on all files for Data Pump export. During an import, you'll need read access to the export dump file. You'll also need write privileges on the directory for import, so that you can write to the log file.

Once you create a directory and grant the necessary rights, all Data Pump export and import jobs can use the `DIRECTORY` parameter to specify the name of the directory object (`DIRECTORY=dpump_dir1`). This way, the `DIRECTORY` parameter will indirectly point to the actual operating system directories and files. Here's an example:

```
$ expdp salapati/password  dumpfile=dpump_dir1.testexp01.dmp
```



You can create a default directory with the name DATA_PUMP_DIR, and then not need to specify the `DIRECTORY` parameter in your export and import commands. Oracle will automatically look for the directory specified as the value for DATA_PUMP_DIR. Data Pump will write all dump files, SQL files, and log files to the directory specified for DATA_DUMP_DIR. Nonprivileged users cannot use this default DATA_PUMP_DIR directory.

Specifying Directory Objects

In order for the Data Pump utilities to know where to place or get data for their export and import jobs, you need to specify location information when you use the `expdp` and `impdp` clients. As you know by now, you can't use absolute directory path location for Data Pump jobs; you must always use a directory object. However, there is more than one way to specify this directory object name during an actual job, as explained in the following sections.

Using the DIRECTORY Parameter Earlier in this section, you learned how to create a directory object. During a Data Pump export job, you can specify the directory object by using the `DIRECTORY` parameter, as shown in the following example.

```
$ expdp hr/hr DIRECTORY=dpump_dir1 …
```

Using the DIRECTORY:FILE Notation You may also specify the directory object without using the `DIRECTORY` parameter. You can do this by specifying the directory object's name as part of the value for a specific Data Pump file (the dump file, log file, or SQL file). You may then use the specific directory object for a log file in the following manner:

```
$ expdp LOGFILE=dpump_dir2:salapati.log …
```

Note that the colon (:) separates the directory and filenames in the log file specification. In this example, dpump_dir2 is the name of the directory object. The Data Pump filename is salapati.log.

Using the DATA_DUMP_DIR Environment Variable You can also use the environment variable `DATA_DUMP_DIR` to point to a file location. In order to use the `DATA_DUMP_DIR` environment, you must have first created a specific directory object on the server. In this example, it is the dpump_dir1 directory described earlier. Once you have this directory, you can then use the `DATA_DUMP_DIR` environment variable on the client to point to the directory object on the server.

In the following example, I first create a new directory object on the server, using the variable `DATA_DUMP_DIR`. I then use the `export` command to save the value of the `DATA_DUMP_DIR` variable in the operating system environment. Once I do that, I can just specify a dump file for my export job, without specifically stating the directory location.

```
SQL> create_directory dump_dir2 AS '/usr/apps/dumpfiles2';
$export  DATA_PUMP_DIR  dump_dir2
$expdp salapati/password TABLES=employees DUMPFILE=employees.dmp
```

Once you have made the DATA_DUMP_DIR variable part of your operating system environment, you don't need to specify the actual directory name (dump_dir2) explicitly (by using the DIRECTORY parameter) when you invoke a Data Pump export, as shown in the previous example. You merely need to specify the *name*, not the *location*, for the DUMPFILE parameter.

Understanding the Order of Precedence for File Locations

Now that we have reviewed the various ways you can specify a directory object for a Data Pump job, you may wonder how Oracle knows which location to use in case there is a conflict. You can have a situation where you muight have specified a DATA_DUMP_DIR environment variable, but you then also specify a DIRECTORY parameter for the export job. Which directory will Oracle choose to use? Here's the order of precedence for directory objects:

1. Oracle will look to see if a directory name is used as part of a file parameter (for example, the LOGFILE parameter). Remember that in these cases, the directory object is separated from the filename by a colon (:).
2. Oracle's second choice would be to use the directory objects assigned to the DIRECTORY parameter during the export or import job. If you explicitly specify the DIRECTORY parameter, you don't need to use the directory name as part of the file parameter.
3. Finally, Oracle looks to see if there is a default server-based directory object named DATA_PUMP_DIR. You must have explicitly created this directory object beforehand. Note that the default DATA_DUMP_DIR object is available only to DBAs and other privileged users.

The directory object name resolution simply means that Oracle knows which directory it should be using to read or write datafiles. However, you must have already granted the database read/write privileges at the operating system level, in order to enable the database to actually use the operating system files.

The Mechanics of a Data Pump Job

The Data Pump export and import utilities use several processes to perform their jobs, including the key master and worker processes, as well as the shadow process and client processes. Let's look at these important Data Pump processes in detail.

The Master Process

The *master process*, or more accurately, the Master Control Process (MCP), has a process name of `DMnn`. The full master process name is of the format `<instance>_DMnn_<pid>`.

There is only one MCP for each job, and this process controls the execution and sequencing of the entire Data Pump job. More specifically, the master process performs the following tasks:

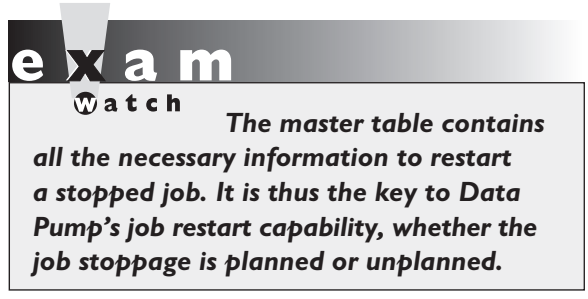
- Creates jobs and controls them
- Creates and manages the worker processes
- Monitors the jobs and logs the progress
- Maintains the job state and restart information in the master table
- Manages the necessary files, including the dump file set

The master process uses a special table called the *master table* to log the location of the various database objects in the export dump file. The master table is at the heart of every Data Pump export and import job. The master process maintains the job state and restart information in the master table. Oracle creates the master table in the schema of the user who is running the Data Pump job at the beginning of every export job. The master table contains various types of information pertaining to the current job, such as the state of the objects in the export/import job, the location of the objects in the dump file set, the parameters of the job, and the status of all worker processes.

The master table has the same name as the export job, such as `SYS_EXPORT_SCHEMA_01`.

The master process uses the master table only for the duration of the export. At the very end of the export, as the last step in the export job, it writes the contents of the master table to the export dump file and automatically deletes the master table from the database. The deletion of the master table will occur automatically, as long as the export completed successfully (or if you issue the `KILL_JOB` command). However, if you use the `STOP_JOB` command to stop a job or the export fails for some reason, the master table isn't deleted from the database. (Data Pump job commands are described in the "Data Pump Export Parameters" section later in this chapter.) When you restart the export job, it will then use the same master table. Since the master table tracks the status of all the objects, Data Pump can easily tell which objects are in the middle of an export and which have been successfully exported to the dump files.

The master process will re-create the master table saved by the export utility in the dump file, in the schema of the user who is performing the import. This is the *first step* in any Data Pump import job. (Note that you don't need to *create* any tables, because the import utility will automatically do this for you.) The Data Pump import utility reads the contents of the master table to verify the correct sequence in which it should import the various exported database objects. As in the case of Data Pump export, if the import job finishes successfully, Oracle will automatically delete the master table.



The Worker Process

The *worker process* is the process that actually performs the heavy-duty work of loading and unloading data, and has the name `DWnn` (`<instance>_DWnn_<pid>`). The MCP (`DMnn`) creates the worker process. The number of worker processes that the master process will create depends on the degree of parallelism of the job. If you choose the `PARALLEL` option for a load, Oracle splits up the worker processes into several parallel execution coordinators.

The worker processes maintain the object rows of the master table. As the worker processes export or import various objects, they update the master table with information about the status of the various jobs: completed, pending, or failed.

Shadow Process

When a client logs in to an Oracle server, the database creates an Oracle foreground process to service Data Pump API requests. This *shadow process* creates the job consisting of the master table as well as the master process. Once a client detaches, the shadow process automatically disappears.

Client Processes

The *client processes* call the Data Pump's API. You perform export and import with the two clients, `expdp` and `impdp`. Later in this chapter, you'll learn about the various parameters you can specify when you invoke these clients.

CERTIFICATION OBJECTIVE 2.02

Using Data Pump Export and Import

The Data Pump export utility corresponds to the traditional export utility, and you invoke it with the client `expdp`. The Data Pump import utility corresponds to the old import utility, and you invoke it with the client `impdp`. In this section, you will learn how to use both Data Pump utilities.

Data Pump export will load row data from database tables as well as object metadata into dump file sets in a proprietary format that only the Data Pump import utility can read. The dump file sets, which are operating system files, will contain data, metadata, and control information. *Dump file sets* usually refer to a single file, such as the default export dump file expdat.dmp.

Quite a few of the Data Pump import utility's features are mirror images of the Data Pump export utility. However, there are some features that are exclusive to the new Data Pump Import utility.

In the following sections, we'll look at Dump Pump export and import types, modes, and parameters, as well as some examples.

Data Pump Export Types

By Data Pump export *types*, I simply mean the various ways in which you can run the Data Pump utility. You can interface with the Data Pump export and import utilities through the command line, using a parameter file, or interactively.

Using the Command Line

You can use the Data Pump export utility from the command line in a manner similar to the traditional export utility. Here's a simple example:

```
$ expdp system/manager directory=dpump_dir1 dumpfile=expdat1.dmp
```

As you can see, the command-line option would quickly get tiring if you were doing anything but the simplest type of exports.

Using a Parameter File

Rather than specifying the export parameters on the command line, you can put them in a parameter file. You then simply invoke the parameter file during the actual export. When you use parameter files, you don't need to retype the same parameters.

For example, you could create a small file called myfile.txt, with the following export parameters:

```
userid=system/manager
directory=dpump_dir1
dumpfile=system1.dmp
```

The file myfile.txt will be your export parameter file. Now, all you need to do in order to export the system schema is invoke `expdp` with just the `PARFILE` parameter, as follows:

```
$ expdp parfile=myfile.txt
```

exam Watch

You can use all command-line export parameters in an export parameter file. The only exception is the parameter PARFILE itself!

Using Interactive Data Pump Export

Since this is a certification upgrade book, I assume you have experience with previous versions of the export and import utilities. You also must be quite familiar with the interactive feature of the export and import utilities. All you need to do during an interactive export or import is merely type **`exp`** or **`imp`** at the command line, and Oracle will prompt you for the rest of the information. Interactive Data Pump export is quite different from the interactive mode of the older utilities. As you'll see in the following sections, Data Pump interactive mode isn't meant to be used in the same way as the `exp`/`imp` interactive mode.

In Data Pump export, you use the interactive method for one purpose only: when you decide you need to change some export parameters midstream, while the job is still running. The way to get into the interactive mode is by pressing the CONTROL-C combination on your keyboard, which interrupts the running job and lets you participate in the export job in an interactive fashion. When you press CONTROL-C during an export job, the running job will pause, and you'll see the export prompt (`Export>`) displayed on your screen. At this point, you can deal interactively with the export utility, with the help of a special set of interesting commands, which I'll explain later in this chapter, in the "Interactive Mode Export Parameters" section. As you'll see, you can also enter the interactive mode of operation by using the `ATTACH` command.

exam Watch

In Data Pump, the interactive mode means that the export or import job stops logging its progress on the screen and displays the export (or import) prompt. You can enter the special interactive commands at this point. Note that the export or import job keeps running throughout, without any interruption.

You can also perform Data Pump export and import operations easily through the OEM Database Control interface. To use this feature, start the Database Control and go to the Maintenance | Utilities page. On that page, you can see the various choices for performing export and import of data.

on the Job

You cannot start an interactive job using Data Pump export (or import). You can use the interactive mode only to intervene during a running job.

Data Pump Export Modes

As in the case of the regular export utilities, you can perform Data Pump export in several modes. The following four modes in which you can perform an export do not differ from the traditional modes of operation using the older export utility:

- **Full export mode** You use the `FULL` parameter when you want to export the entire database in one export session. You need the `EXPORT_FULL_DATABASE` role to use this mode.
- **Schema mode** If you want to export a single user's data and/or objects only, you must use the `SCHEMA` parameter.
- **Tablespace mode** By using the `TABLESPACES` parameter, you can export all the tables in one or more tablespaces. If you use the `TRANSPORT_TABLESPACES` parameter, you can export just the metadata of the objects contained in one or more tablespaces. You may recall that you can export tablespaces between databases by first exporting the metadata, copying the files of the tablespace to the target server, and then importing the metadata into the target database.
- **Table mode** By using the `TABLES` parameter, you can export one or tables. The `TABLES` parameter is identical to the `TABLES` parameter in previous versions of Oracle.

Data Pump Export Parameters

Some of the Data Pump export commands are familiar to you from the traditional export utility. Others are quite new. Here, I'll briefly run through the set of Data Pump export parameters, providing detailed explanations for only the new and unfamiliar parameters. For this discussion, the parameters are grouped into the following categories:

- File- and directory-related parameters
- Export mode-related parameters
- Export filtering parameters
- Estimation parameters
- The network link parameter
- Interactive mode export parameters
- Job-related parameters

You can use all the following parameters at the command line or in parameter files, except those listed in the "Interactive Mode Export Parameters" section.

File- and Directory-Related Parameters

You can specify several file- and directory-related parameters during a Data Pump export job. Let's look at these parameters in the following sections.

DIRECTORY The `DIRECTORY` parameter specifies the location of the dump and other files. A detailed discussion of how you can use this parameter was presented in the "Using Directory Objects" section earlier in this chapter.

DUMPFILE The `DUMPFILE` parameter provides the name of the dump file to which the export dump should be written. The `DUMPFILE` parameter replaces the `FILE` parameter in the old export utility. You can provide multiple dump filenames in several ways:

- You can create multiple dump files by specifying the `%U` substitution variable.
- You can provide multiple files in a comma-separated list.
- You can specify the `DUMPFILE` parameter multiple times for a single export job.

If you specify the %U notation to indicate multiple dump files, the number of files you can create is equal to the value of the PARALLEL parameter.

If you don't specify the DUMPFILE parameter, Oracle will use the default name expdat.dmp for the export dump file, just as it does when you use the traditional export utility.

FILESIZE The FILESIZE parameter is purely optional, and it specifies the size of the dump file. If you don't specify this parameter, the dump file has no limits on its size. If you use the FILESIZE parameter by specifying, say 10MB as the maximum dump file size, your export will stop if your dump file reaches its size limit, and you can restart it after correcting the problem.

PARFILE The PARFILE parameter stands for the same thing it did in traditional export utility: the parameter file, wherein you can specify export parameters in a file, instead of entering them directly from the command line.

LOGFILE and NOLOGFILE You can use the LOGFLE parameter to specify a log file for your export jobs. Here's what you need to remember regarding this parameter:

If you just specify the parameter without the directory parameter, Oracle will automatically create the log file in the location you specified for the DIRECTORY parameter.

- If you don't specify this parameter, Oracle will create a log file named export.log. A subsequent export job will overwrite this file, because Oracle always names the default log file simply *export.log*.
- If you specify the parameter NOLOGFILE, Oracle will not create its log file (export.log). You'll still see the progress of the export job on the screen, but Oracle suppresses the writing of a separate log file for the job.

Export Mode-Related Parameters

The export mode-related parameters are the FULL, SCHEMAS, TABLES, TABLESPACES, TRANSPORT_TABLESPACES, and TRANSPORT_FULL_CHECK parameters. You've already seen all these parameters except the last one, in the "Data

Pump Export Modes" section. The TRANSPORT_FULL_CHECK parameter simply checks to make sure that the tablespaces you are trying to transport meet all the conditions to qualify for the job.

Export Filtering Parameters

There are several new parameters related to export filtering. Some of them are substitutes for old export parameters, and others offer new functionality. Let's look at these important parameters in detail.

CONTENT By using the CONTENT parameter, you can filter what goes into the export dump file. The CONTENT parameter can take three values:

- ALL exports both table data and table and other object definitions (metadata).
- DATA_ONLY exports only table rows.
- METADATA_ONLY exports only metadata.

Here's an example:

```
$ expdp system/manager dumpfile=expdat1.dmp content=data_only
```

Note that the CONTENT=METADATA_ONLY option is equivalent to the rows=*n* option in the original export utility. However, there is no equivalent to the CONTENT=DATA_ONLY option in Data Pump.

EXCLUDE and INCLUDE The EXCLUDE and INCLUDE parameters are two *mutually exclusive parameters* that you can use to filter various kinds of objects. Remember how in the old export utility you used the CONSTRAINTS, INDEXES, GRANTS, and INDEXES parameters to specify whether you wanted to export those objects? Using the EXCLUDE and INCLUDE parameters, you now can include or exclude many other kinds of objects besides the four objects you could previously.

Simply put, the EXCLUDE parameter helps you *omit* specific database object types from an export or import operation. The INCLUDE parameter, on the other hand, enables you to *include* only a specific set of objects. Following is the format of the EXCLUDE and INCLUDE parameters:

```
EXCLUDE=object_type[:name_clause]
INCLUDE=object_type[:name_clause]
```

For both the EXCLUDE and INCLUDE parameters, the name clause is optional. As you know, several objects in a database—such as tables, indexes, packages, and procedures—have names. Other objects, such as grants, don't have names. The name clause in an EXCLUDE or an INCLUDE parameter lets you apply a SQL function to filter named objects.

Here's a simple example that excludes all tables that start with *EMP*:

```
EXCLUDE=TABLE:"LIKE 'EMP%'"
```

In this example, `"LIKE 'EMP%'"` is the name clause.

The name clause in an EXCLUDE or INCLUDE parameter is optional. It's purely a filtering device, allowing you finer selectivity within an object type (index, table, and so on). If you leave out the name clause component, all objects of the specified type will be excluded or included.

In the following example, Oracle excludes all indexes from the export job, since there is no name clause to filter out only some of the indexes.

```
EXCLUDE=INDEX
```

You can also use the EXCLUDE parameter to exclude an entire schema, as shown in the following example.

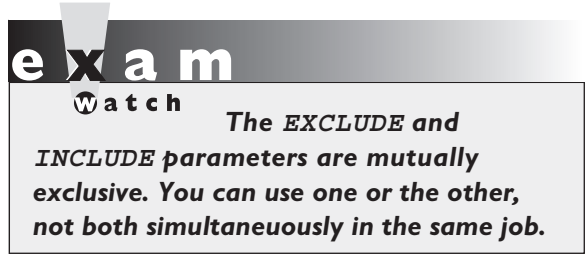
```
EXCLUDE=SCHEMA:"='HR'"
```

The INCLUDE parameter is the precise opposite of the EXCLUDE parameter: it forces the *inclusion* of only a set of specified objects in an export. As in the case of the EXCLUDE parameter, you can use a name clause to qualify exactly which objects you want to export. Thus, you have the ability to selectively choose objects at a fine-grained level.

The following three examples show how you can use the name clause to limit the selection of objects.

```
INCLUDE=TABLE:"IN ('EMPLOYEES', 'DEPARTMENTS')"
INCLUDE=PROCEDURE
INCLUDE=INDEX:"LIKE 'EMP%'"
```

The first example is telling the Data Pump job to only include two tables: employees and departments. In the second example, the INCLUDE parameter specifies that only procedures should be included in this export job. The third example shows how you can specify that only those indexes that start with *EMP* should be part of the export job.



QUERY The QUERY parameter stands for the same thing as it does in traditional export: it lets you selectively export table row data with the help of a SQL statement. However, the parameter is enhanced for Oracle Database 10*g* by permitting you to qualify the SQL statement with a table name, so that it applies only to a particular table. Here's an example:

```
QUERY=OE.ORDERS: "WHERE order_id > 100000"
```

In this example, only those rows in the orders table where the order_id is greater than 100000 are exported.

Estimation Parameters

Two interesting parameters enable you to estimate how much physical space your export job will consume. Let's look at both these parameters in detail.

ESTIMATE The ESTIMATE parameter will tell you how much space your new export job is going to consume. The space estimate is always in terms of bytes. By default, Oracle will always estimate the space requirements in terms of *blocks*. It simply takes your database block size and multiplies it with the amount of blocks all the objects together will need. Here is an example of what you'll see in your log file (and on the screen):

```
Estimate in progress using BLOCKS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 654 KB
```

Since the space estimation in terms of blocks is the default behavior, you don't need to specify the ESTIMATE parameter during the export. However, if you have analyzed all your tables recently, you can ask the Data Pump export utility to

estimate the space requirements by using the statistics the database has already calculated for each of the tables. In order to tell the database to use the database statistics (rather than use the default BLOCKS method), you need to specify the ESTIMATE parameter in the following manner:

```
ESTIMATE=statistics
```

Here's what you'll see in your log file when you use the ESTIMATE=statistics parameter:

```
Estimate in progress using STATISTICS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
.  estimated "SYSTEM"."HELP"                                35.32 KB
Total estimation using STATISTICS method: 65.72 KB
```

ESTIMATE_ONLY While the ESTIMATE parameter is operative only during an actual export job, you can use the ESTIMATE_ONLY parameter *without* starting an actual export job. Here's an example:

```
C:\>expdp system/manager estimate_only=y
Export: Release 10.1.0.2.0 - Production on Saturday, 17 April, 2004 14:30
…
Total estimation using BLOCKS method: 288 KB
Job "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully completed at 14:30
```

Although the log indicates that the export job "completed successfully," all the job really did was to estimate the space that you will need for the export job.

The Network Link Parameter

The expdp utility provides you with a way to initiate a *network export*. Using the NETWORK_LINK parameter, you can initiate an export job from your server and have Data Pump export data from a remote database to dump files located on the instance from which you initiate the Data Pump export job.

Here's an example that shows you how to perform a network export:

```
expdp hr/hr DIRECTORY=dpump_dir1 NETWORK_LINK=finance@prod1
DUMPFILE=network_export.dmp LOGFILE=network_export.log
```

In the example, the NETWORK_LINK parameter must have a valid database link as its value. This means that you must have created the database link ahead of time. You are exporting data from the finance database on the prod1 server.

on the Job

You can't use Data Pump in the normal way to export data from a read-only database. This is because Data Pump can't create the necessary master table or create external tables on a read-only tablespace. Using the network mode, however, you can export data from a read-only database on server A to dump files on server B, where Data Pump is running.

Interactive Mode Export Parameters

As I mentioned earlier in this chapter, the interactive mode of Data Pump is quite different from the interactive export and import mode that you know. Traditionally, the interactive mode gave you the chance to enter a limited set of export/import parameters at the command line in response to the queries made by the export or import utility. You use the interactive mode in the new Data Pump technology only to intervene in the middle of a running job, to either suspend the job or modify some aspects of it. You can enter the interactive mode of Data Pump export in either of two ways:

- You can use the CONTROL-C keyboard combination during a Data Pump export job, if you want to enter the interactive mode from the same session where you are running the Data Pump job.
- You can either use a separate session or even a separate server to "attach" yourself to a running session by using—what else?—the ATTACH command. (You can also attach to a stopped job.) When you successfully attach yourself to a job, you'll be able to use specific export parameters in an interactive mode.

exam Watch

In the Data Pump export (and import), the only way to get into an interactive mode of operation is by using the CONTROL-C sequence or by opening another session and "attaching" yourself to that session. You cannot start an interactive Data Pump session from the command line.

Let's examine when you might use the interactive mode in Data Pump export. Suppose that you started a job in the evening at work and left for home. At midnight, you check the status of the job and find that it's barely moving. You can easily start another session, and then "connect" to the running job and monitor it by simply using the ATTACH command. When you do this, the running job does not pause.

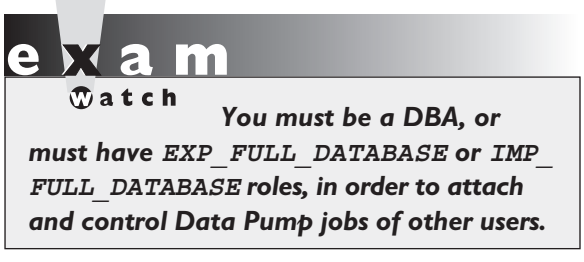
Instead, it opens an interactive window into the running session, so you can change some parameters to hasten the crawling export job by running one of a special set of interactive Data Pump export commands. Here's an example of the usage of the ATTACH parameter:

```
C:\>expdp salapati/sammyy1 attach=SALAPATI.SYS_EXPORT_SCHEMA_01
Export: Release 10.1.0.2.0 - Production on Saturday, 17 April, 2004 11:47
…
State: EXECUTING
…
Export>
```

You may attach multiple clients to a single job.

Once you attach yourself to a running job by using the ATTACH command or by using the CONTROL-C sequence on the server where the job is actually running, you get the interactive export prompt (Export>), indicating that Data Pump is awaiting your interactive commands. From the interactive prompt, you can use several interesting parameters to influence the progress of the currently executing Data Pump job. Here are some examples of interactive commands:

```
Export> parallel=4
Export> kill_job
Export> stop_job
Export> continue_client
```



I'll explain these and other interactive Data Pump parameters in the following sections, grouped in the categories of client-related parameters, job-related parameters, and other parameters.

Client-Related Interactive Parameters The CONTINUE_CLIENT parameter will take you out of the interactive mode and resume the running export job. Your client connection will still be intact, and you'll continue to see the export messages on your screen. However, the EXIT_CLIENT parameter will stop the interactive session, as well as terminate the client session. In both of these cases, the actual Data Pump export job will continue to run unhindered.

Job-Related Interactive Parameters You can use several job-related parameters from any interactive session you open with an export session using the `ATTACH` command. You can use the `STOP_JOB` command to stop the export job in an orderly fashion. To stop it at once, use the `STOP_JOB=immediate` command. You can choose to resume any export jobs you've stopped in this manner, with the help of the `START_JOB` parameter.

If you decide that you don't really want to continue the job you've just attached to, you can terminate it by using the `KILL_JOB` parameter. Unlike the `EXIT_CLIENT` parameter, the `KILL_JOB` parameter terminates *the client as well as the export job* itself for good.

To summarize, the job-related interactive parameters work as follows:

- `STOP_JOB` stops running Data Pump jobs.
- `START_JOB` resumes stopped jobs.
- `KILL_JOB` kills both the client and the Data Pump job.

exam Watch

You can restart any job that is stopped, whether it's stopped because you issued a `STOP_JOB` command or due to a system crash, as long as you have access to the master table and an uncorrupted dump file set.

Other Interactive Parameters From the interactive prompt, you can use the `ADD_FILE` parameter to add a dump file to your job. You can also use the `HELP` and `STATUS` parameters interactively, and both of these parameters function the same way as their command-line counterparts.

Job-Related Parameters

Several Data Pump export parameters can be classified as job-related parameters. I'll briefly discuss the important ones here.

JOBNAME You can use the `JOBNAME` parameter to provide your own job name for a given Data Pump export/import job (for example, `JOBNAME=myjob1`). The `JOBNAME` parameter is purely optional, however. If you don't use it, Data Pump will generate a unique system name, of the format `<USER>_<OPERATION>_<MODE>_%N`. For example, if the user SYSTEM is performing an export of the

database (FULL mode), the automatically generated job name will be SYSTEM_EXPORT_FULL_01. In this job name, SYSTEM is the user that is performing the Data Pump job. EXPORT tells you it's an export, not an import job. FULL indicates that this is full database export. The last part of the job name is a number, indicating the sequence number of the job. This was my first job in a new database, so my job number happens to end with 01.

Remember that Oracle gives the master table, which holds critical information about your export job, the same name as the name of the job.

STATUS The STATUS parameter is useful while you're running long jobs, as it provides you with an updated status at intervals that you can specify. The parameter takes integer values that stand for seconds. For example, an anxious DBA (like me) might want to get an update every minute regarding an ongoing Data Pump export job. Here's what you need to do to get your reassuring minutely updates:

```
$ expdp system/manager status=60 …
…
Worker 1 Status:
  State: EXECUTING
  Object Schema: SYSTEM
  Object Name: SYS_EXPORT_SCHEMA_01
  Object Type: SCHEMA_EXPORT/TABLE/TABLE_DATA
  Completed Objects: 1
  Total Objects: 65
. . exported "SYSTEM"."REPCAT$_SITES_NEW"
Job: SYS_EXPORT_SCHEMA_01
  Operation: EXPORT
  Mode: SCHEMA
  State: EXECUTING
  Bytes Processed: 69,312
  Percent Done: 99
  Current Parallelism: 1
  Job Error Count: 0
  Dump File: C:\ORACLE\PRODUCT\10.1.0\ADMIN\EXPORT\EXPDAT6.DMP
    bytes written: 1,748,992
…
```

The STATUS parameter shows the overall percentage of the job that is completed, the status of the worker processes, and the status of the current data objects being processed.

PARALLEL PARALLEL is the mighty parameter that lets you specify more than a single active execution thread for your export job. Note that the Data Pump

PARALLEL parameter has nothing to do with the other Oracle parallel features, but they can work together. The default value of the PARALLEL parameter is 1, meaning a single thread export operation writing to a single dump file. If you specify anything more than 1 as the value for the PARALLEL parameter, you also should remember to specify the same number of dump files, so the multiple execution threads can simultaneously write to the multiple dump files. Here's an example that shows how you can set the level of parallelism to 3, forcing the export job to write in parallel to three dump files:

```
expdp system/manager DIRECTORY=dpump_dir1 DUMPFILE=par_exp%u.dmp PARALLEL=3
```

If you specify the PARALLEL parameter, make sure you allocate the same number of dump files as the degree of parallelism. The higher the degree of parallelism, the higher will be the memory, CPU, and network bandwith usage as well.

In the previous example, the DUMPFILE parameter uses the substittion varable %u to indicate that multiple files should be generated, of the format par_exp***NN***.*dmp*, where *NN* is a two-character integer starting with 01. Since the PARALLEL parameter is set to 3, the substitution variable will create three files with the following names: par_exp01.dmp, par_exp02.dmp, and par_exp03.dmp.

Note that you don't need to use the %u substition variable to generate multiple dump files when you choose a value of greater than 1 for the PARALLEL parameter. You could simply use a comma-separated list of values, as follows:

```
expdp system/manager DIRECTORY=dpump_dir1 PARALLEL 3
DUMPFILE=(par_exp01.dmp,par_exp02.dmp,par_exp03.dmp)
```

If you don't have sufficient I/O bandwidth, you may actually experience a degradation in Data Pump performance with the PARALLEL parameter.

Data Pump Export Examples

Let's look at some simple Data Pump export job specifications that demonstrate some of the new concepts you've learned in this chapter. The next example creates an export dump file of just two tables: employees and jobs.

```
expdp hr/hr TABLES=employees,jobs DUMPFILE=dpump_dir1:table.dmp NOLOGFILE=y
```

The following example shows how to use a parameter file, as well as how to use the CONTENT and EXCLUDE parameters. The CONTENT=DATA_ONLY specification means you are exporting just rows of data and excluding all object definitions (metadata). The EXCLUDE parameter requries that the countries, locations, and regions tables be omitted from the export. The QUERY parameter stipulates that all the data in the employees table, except that belonging to location_id 20, be exported. The parameter file, exp.par, has the following information:

```
DIRECTORY=dpump_dir1
DUMPFILE=dataonly.dmp
CONTENT=DATA_ONLY
EXCLUDE=TABLE:"IN ('COUNTRIES', 'LOCATIONS', 'REGIONS')"
QUERY=employees:"WHERE department_id !=20 ORDER BY employee_id"
```

You can then issue the following command to execute the `exp.par` parameter file:

```
$ expdp hr/hr PARFILE=exp.par
```

The following example illustrates a schema mode export. You don't see any mention of the SCHEMA parameter; that's because Data Pump will export a schema (of the exporting user) by default.

```
$ expdp hr/hr DUMPFILE=dpump_dir1:expschema.dmp
LOGFILE=dpump_dir1:expschema.log
```

By default, Data Pump export will run the export in the schema mode.

Here's an interesting Data Pump export example, showing how to use the PARALLEL, FILESIZE, and JOB_NAME parameters. It also illustrates the use of the DUMPFILE parameter when there are multiple dump files.

```
$ expdp hr/hr FULL=y DUMPFILE=dpump_dir1:full1%U.dmp, dpump_dir2:full2%U.dmp
FILESIZE=2G PARALLEL=3 LOGFILE=dpump_dir1:expfull.log JOB_NAME=expfull
```

Now that you've seen how the Data Pump export utility works, you're ready to look at the Data Pump import features.

Data Pump Import Types and Modes

As in the case of exporting data, you can perform a Data Pump import job from the command line or use a parameter file. Interactive access to the import utility is available, but it is different from what you are used to when working with the

traditional export/import utilities. The interactive framework is analogous to the interactive access to the Data Pump export utility, as you'll see shortly.

You can use Data Pump import in the same modes as Data Pump export: table, schema, tablespace, and full modes. In addition, you can also employ the TRANSPORTABLE_TABLESPACES parameter to import the metadata necessary for implementing the transportable tablespaces feature.

You must have the IMPORT_FULL_DATABASE role in order to perform one of the following:

- Full database import
- Import of a schema other than your own
- Import of a table that you don't own

exam Watch

You'll need the IMPORT_FULL_DATABASE role to perform an import if the dump file for the import was created using the EXPORT_FULL_DATABASE role.

Data Pump Import Parameters

As in the case of the Data Pump export utility, you control a Data Pump import job with the help of several parameters when you invoke the impdp utility. For this discussion, the import parameters are grouped as follows:

- File- and directory-related parameters
- Filtering parameters
- Job-related parameters
- Import mode-related parameters
- Remapping parameters
- The network link parameter
- The transform parameter
- The flashback time parameter

File- and Directory-Related Parameters

The Data Pump import utility uses the PARFILE, DIRECTORY, DUMPFILE, LOGFILE, and NOLOGFILE commands in the same way as the Data Pump export utility. However, SQLFILE is a file-related parameter unique to the import utility.

The SQLFILE parameter is similar to the old import utility's INDEXFILE parameter. When you perform a Data Pump import, you may sometimes wish to

extract the DDL from the export dump file. The SQLFILE parameter enables you to do this easily, as shown in the following example.

```
$ impdp salapati/sammyy1 DIRECTORY=dpump_dir1 DUMPFILE=finance.dmp
SQLFILE=dpump_dir2:finance.sql
```

In this example, the SQLFILE parameter instructs the Data Pump import job to write the DDL to the finance.sql file, located in the directory dpump_dir2. Of course, you must have created dpump_dir2 prior to this, using the CREATE DIRECTORY AS command. The DIRECTORY=dpump_dir1 parameter value tells Data Pump import where to find the dump file finance.dmp. This example also shows how you can use multiple directories in a single Data Pump job.

It's important to remember that the SQLFILE parameter just *extracts* the SQL DDL to the specified file—no actual data import whatsoever takes place. By using this parameter, you can extract a SQL script with all the DDL from your export dump file. The DDL in SQLFILE lets you peek at what the import job will execute.

The other import file-related parameter is the new REUSE_DATAFILES parameter. This parameter tells Data Pump whether it should use existing datafiles for creating tablespaces during an import. If you specify REUSE_DATAFILES=y, the import utility will write over your existing datafiles.

Filtering Parameters

You use the CONTENT parameter, as in the case of a Data Pump export, to determine whether you'll load just rows (CONTENT=DATA_ONLY), rows and metadata (CONTENT=ALL), or just metadata (CONTENT=METADATA_ONLY).

The EXCLUDE and INCLUDE parameters have the same meaning as in an export, and they are mutually exclusive. If you use the CONTENT=DATA_ONLY option, you cannot use either the EXCLUDE or INCLUDE parameter during an import.

You can use the QUERY parameter during import as well, in order to filter data during an import. In the older export/import utilities, you could use the QUERY parameter only during an export. You can use the QUERY parameter to specify an entire schema or a single table. Note that if you use the QUERY parameter during import, Data Pump will use only the external table data method, rather than the direct-path method, to access the data.

What will Data Pump import do if there is a table creation script in the export dump file, but the *table already exists* in the target database? You can use the TABLE_EXISTS_ACTION parameter to tell Data Pump what to do when a table

already exists. You can provide four different values to the TABLE_EXISTS_ACTION parameter:

- With SKIP (the default), Data Pump will skip a table if it exists.
- The APPEND value appends rows to the table.
- The TRUNCATE value truncates the table and reloads the data from the export dump file.
- The REPLACE value drops the table if it exists, re-creates, and reloads it.

Job-Related Parameters

The JOB_NAME, STATUS, and PARALLEL parameters carry identical meanings as their Data Pump export counterparts. Note that if you have multiple dump files, you should specify them either explicitly or by using the %u notation, as shown in the Data Pump import section.

Import Mode-Related Parameters

You can perform a Data Pump import in various modes, using the TABLE, SCHEMAS, TABLESPACES, and FULL parameters, just as in the case of the Data Pump export utility. You can use the TRANSPORTABLE_TABLESPACES parameter when you wish to transport tablespaces between databases.

Remapping Parameters

The remapping parameters are brand-new features in the Oracle Database 10*g* Data Pump import utility, and they clearly mark the superiority of this utility over the traditional import utility. Let's briefly discuss each of these three parameters: REMAP_SCHEMA, REMAP_DATAFILE, and REMAP_TABLESPACE.

REMAP_SCHEMA Using the REMAP_SCHEMA parameter, you can move objects from one schema to another. You need to specify this parameter in the following manner:

```
$ impdp system/manager dumpfile=newdump.dmp REMAP_SCHEMA=hr:oe
```

In this example, HR is the source schema, and Data Pump import will import all of user HR's objects into the target schema OE. The import utility can even create the OE schema, if it doesn't already exist in the target database. Of course, if you want to

just import one or more tables from the HR schema and import them into the OE schema, you can do that as well, by using the TABLES parameter.

The REMAP_SCHEMA parameter provides the same functionality as the FROMUSER/TOUSER capability in the old export and import utilities.

REMAP_DATAFILE When you are moving databases between two different platforms, each with a separate filenaming convention, the REMAP_DATAFILE parameter comes in handy to change file system names. The following is an example that shows how you can change the file system from the old Windows platform to the new UNIX platform. Whenever there is any reference to the Windows file system in the export dump file, the import utility will automatically remap the filename to the UNIX file system.

```
$ impdp hr/hr FULL=y DIRECTORY=dpump_dir1 DUMPFILE=db_full.dmp
REMAP_DATAFILE='DB1$:[HRDATA.PAYROLL]tbs6.f':'/db1/hrdata/payroll/tbs6.f'
```

REMAP_TABLESPACE Sometimes, you may want the tablespace into which you are importing data to be different from the tablespace in the source database. The REMAP_TABLESPACE parameter enables you to move objects from one tablespace into a different tablespace during an import, as shown in the following example. Here, Data Pump import is transferring all objects from the tablespace example_tbs to the tablespace new_tbs.

```
$ impdp hr/hr REMAP_TABLESPACE='example_tbs':'new_tbs'
DIRECTORY=dpump_dir1
PARALLEL=2 JOB_NAME=cf1n02 DUMPFILE=employees.dmp NOLOGFILE=Y
```

The Network Link Parameter

Using the new NETWORK_LINK parameter, you can perform an import across the network, without using dump files. The NETWORK_LINK parameter enables import to connect directly to the source database and transfer data to the target database. Here's an example:

```
$ impdp hr/hr TABLES=employees DIRECTORY=dpump_dir1
  NETWORK_LINK=finance@prod1 EXCLUDE=CONSTRAINT
```

In this example, finance@prod1 is the network link. It is a valid database link, created by you beforehand using the CREATE DATABASE LINK command. Thus, the database shown in the database link is your source for the import job. Data Pump will import

the table employees from the remote database finance to your instance where you run the Data Pump import job. In a network import, the Metadata API executes on the remote instance, and extracts object definitions and re-creates necessary objects in your local instance. It then fetches data from the remote database tables and loads them in your local instance, using the `INSERT AS SELECT` command, as follows:

```
insert into employees(emp_name,emp_id) … select (emp_name,emp_id) from
finance@remote_service_name
```

Note that a Data Pump network import *doesn't involve a dump file*, as Data Pump will import the table from the source to the target database directly.

EXERCISE 2-1

Using the NETWORK_LINK Parameter

Using the following information as your guidelines, perform an an import using the `NETWORK_LINK` parameter.

```
SQL> create database link L1
     connect to system identified by oracle
     using 'db_name';
SQL> create directory d1 as 'e:\tmp';
E:\> impdp userid=system/oracle tables=hr.regions  network_link=L1
remap_schema=HR:OE directory=D1
```

The TRANSFORM Parameter

Suppose you are importing a table from a different schema or even a different database. Let's say you want to make sure that you don't also import the objects' storage attrributes during the import—you just want to bring in the data that the table contains. What can you do? The new `TRANSFORM` parameter lets you specify that your Data Pump import job should not import certain storage and other attributes. Using the `TRANSFORM` parameter, you can exclude the `STORAGE` and `TABLESPACE` clauses, or just the `storage` clause, from a table or an index.

During a Data Pump (or traditional) import, Oracle creates objects using the DDL that it finds in the export dump files. The `TRANSFORM` parameter instructs the Data Pump import job to modify the DDL that creates the objects during the import job.

The TRANSFORM parameter has the following syntax:

```
TRANSFORM = transform_name:value[:object_type]
```

Here's an example to help you understand the TRANSFORM parameter:

```
impdp hr/hr TABLES=hr.employees \
  DIRECTORY=dpump_dir1 DUMPFILE=hr_emp.dmp \
  TRANSFORM=SEGMENT_ATTRIBUTES:n:table
```

The TRANSFORM parameter syntax elements correspond to the following items in the example:

- **Transform name** You can modify two basic types of an object's characteristics using TRANSFORM: *segment attributes* and *storage*. Segment attributes include physical atttributes, storage attributes, tablespaces, and logging. The transform name represents exactly which of these two object attributes you want to modify during the import job. In the example, the TRANSFORM=SEGMENT_ATTRIBUTES specification indicates that you want the import job to modify all the segment (the employees table in the HR schema) attributes.
- **Value** The value of the TRANSFORM parameter can be Y (yes) or N (no). By default, the value is set to Y. This means that, by default, Data Pump imports an object's segment attributes and storage features. If you assign a value of N as your choice, you specify not to import the original segment attributes and/or the storage attributes.
- **Object type** The object type specifies which types of objects should be transformed. Your choices are limited to TABLE and INDEX. You may omit this part of the TRANSFORM parameter specification, in which case Data Pump import will transform both tables and indexes.

The Flashback Time Parameter

The FLASHBACK_TIME parameter enables you to import data consistent as of the flashback time you specify in your import job. For example, look at the following import statement:

```
$ impdp system/manager flashback_time=2004-06-01  07:00
```

The import job will ensure that the data is consistent as of the time you specified. Note that the FLASHBACK_TIME parameter does the same thing as the old CONSISTENT parameter in the traditional import utility.

CERTIFICATION OBJECTIVE 2.03

Monitoring a Data Pump Job

There are two new views—DBA_DATA PUMP_JOBS and DBA_DATA PUMP_SESSIONS—which are crucial for monitoring Data Pump jobs. In addition, you can also use the V$SESSION_ LONGOPS view and the old standby V$SESSION, to obtain session information. In most cases, you can join two or more of these views to gain the necessary information about job progress. Let's look at some of the important data dictionary views that help you manage Data Pump jobs.

Viewing Data Pump Jobs

The DBA_DATA PUMP_JOBS view shows summary information of all currently running Data Pump jobs. It has the following structure:

```
SQL> desc dba_data pump_jobs
 Name                  Null?        Type
 ----------------------------------------------
 OWNER_NAME                         VARCHAR2(30)
 JOB_NAME                           VARCHAR2(30)
 OPERATION                          VARCHAR2(30)
 JOB_MODE                           VARCHAR2(30)
 STATE                              VARCHAR2(30)
 DEGREE                             NUMBER
 ATTACHED_SESSIONS                  NUMBER
```

Since the dynamic DBA_DATA PUMP_JOBS view shows only the active jobs, you can easily find the JOB_NAME value for any job that is running right now. As you know, you'll need to know the job name for a job if you want to attach to a running job in midstream.

Because the name of the master table is the same as the JOB_NAME value, you can thus determine the name of the master table through this view.

The JOB_MODE column can take the values FULL, TABLE, SCHEMA, or TABLESPACE, reflecting the mode of the curently executing export or the import job.

The STATE column can take the values UNDEFINED, DEFINING, EXECUTING, and NOT RUNNING, depending on which stage of the export or import you execute your query. The Data Pump job enters the NOT RUNNING state immediately before it completes the import or export. Of course, when there aren't any active jobs running, the view DBA_DATA PUMP_JOBS returns no rows whatsoever.

Viewing Data Pump Sessions

The DBA_DATA PUMP_SESSIONS view identifies the user sessions currently attached to a Data Pump export or import job. You can join the SADDR column in this view with the SADDR column in the V$SESSION view to gain useful information about user sessions that are currently attached to a job. The following query shows this:

```
SQL> select sid, serial#
     from v$session s, dba_data pump_sessions d
     where s.saddr = d.saddr;
```

Viewing Data Pump Job Progress

The V$SESSION_LONGOPS dynamic performance view is not new to Oracle Database 10*g*. In Oracle 9*i*, you could use use this view to monitor long-running sessions.

In the V$SESSION_LONGOPS view, you can use the columns TOTALWORK, SOFAR, UNITS, and OPNAME to monitor the progress of an export/import job. This is what these four key columns represent:

- TOTALWORK shows the total estimated number of megabytes in the job.
- SOFAR shows the megabytes transferred thus far in the job,
- UNITS stands for megabytes.
- OPNAME shows the Data Pump job name.

Here's a typical SQL script that you can run to show how much longer it will take for your Data Pump job to finish:

```
SQL> select sid, serial#, sofar, totalwork
     from v$session_longops
     where opname = 'MY_EXPORTJOB1'
```

CERTIFICATION OBJECTIVE 2.04

Creating External Tables for Data Population

External tables are tables that do not reside in the database itself, and they can have any format defined by an access driver. An external table is merely a representation of external data in a file—data that's never actually loaded into an Oracle table. In some

ways, external tables are like a view, but the data physically exists in a flat file outside the database.

External tables aren't a new feature of Oracle Database 10*g*. Oracle9*i* first introduced the concept of external tables. However, in Oracle9*i*, you could only read from external tables. Now, in Oracle Database 10*g*, you can also to *write to* external tables.

We'll look at the following areas regarding external table creation in the following sections:

- An overview of external table population features
- The process for creating external tables
- How to load and unload data
- Parallel population of external tables

Features of External Table Population Operations

In reality, an external table isn't really a table, but rather an interface to an external datafile. However, you may query this external table like a virtual table, just as you would query any regular Oracle table, which makes it a very powerful tool for data warehouse extraction, transformation, and loading (ETL) activities. You can query external tables or join them with regular tables, without ever loading the external data into your database. In addition, you may create other regular tables or views from the external tables, so this feature comes in very handy during the population of data warehouses.

You can't perform all the normal table Data Manipulation Language (DML) actions on an external table. You can query the external table data, but you can't perform an update, delete, or an insert on an external table. You also can't build an index on external tables.

Prior to Oracle Database 10*g*, you could use external tables to load data into a database from external flat files. Now, for the first time, you can unload data from an Oracle database into an external table. That is, whereas you could only *read* from an external table before, now you can *write* to an external table as well. The technique simply uses the `CREATE TABLE AS SELECT` (CTAS) command to populate external tables with data that actually resides in operating system *text files* and not in regular *datafiles*.

exam Watch

The same limitations in earlier versions—inability to create indexes and perform DML on the external tables—still apply to all external tables.

When you create an external table, you can use the TYPE attribute to select between two types of external tables: the ORACLE_LOADER type and the ORACLE_DATAPUMP type. Each of these external tables comes with its own access driver. In Oracle9*i*, you used the ORACLE_LOADER access driver to create external tables; however, the ORACLE_LOADER access driver can load data only into an external table; that is, it can extract data from external flat files to load an Oracle (external) table. The ORACLE_LOADER access driver is the default access driver in Oracle Database 10*g*.

The ORACLE_DATAPUMP access driver is new to Oracle Database 10*g*. The ORACLE_DATA PUMP access driver can load as well extract data; that is, it can both load an external table from a flat file and extract data from a regular database table to an external flat file. This external flat file data is written in a proprietary format, which only the ORACLE_DATA PUMP access driver can read. You can then use this newly created file to create an external table in the same database or a different database.

exam Watch

If you want to create indexes on a staging table, you are better off using the SQL*Loader utility to load data into the table. You cannot index an external table!

Here's a summary of the main features of external table population operations:

- You can use the ORACLE_LOADER or ORACLE_DATA PUMP access drivers to perform data loads. You can use only the new ORACLE_DATA PUMP access driver for unloading data (populating external tables).
- No DML or indexes are possible for external tables.
- You can use the datafiles created for an external table in the same database or a different database.

exam Watch

The new Oracle Database 10g ORACLE_DATA PUMP access driver can perform a data load as well as an unload. The older ORACLE_LOADER access driver can only load an external table using data in operating system text files.

Creating External Tables

In this section, I'll briefly describe the mechanics of creating an external table. This basic background will help you to understand the enhancements in this area in Oracle Database 10*g*. The three main steps—create the datafile, create the directory object, and then create the external table—are demonstrated in the following sections.

Create the Datafile

Create a flat file with some data that you'll load into your external table. Let's call this datafile dept.dmp. Later, you'll be defining this flat file as an external table. The file will always remain in the operating system directories, and you may edit it as you wish. Here's the datafile structure:

```
10000001,nina,FINANCE,04-APR-2000
10000002,nicholas,FINANCE,04-APR-2000
10000007,shannon,HR,02-FEB-1990
10000008,valerie,HR,01-JUN-1998
```

Create the Directory Object

Create a directory object to hold the external datafiles, as shown here:

```
SQL> CREATE OR REPLACE DIRECTORY employee_data  AS 'C:\employee_data';
Directory created.
```

Create the External Table

Use the CREATE TABLE ... ORGANIZATION EXTERNAL statement to create your external table, as follows:

```
SQL> CREATE TABLE employee_ext
          (
           empid  NUMBER(8),
           emp_name  VARCHAR2(30),
           dept_name VARCHAR2(20),
           hire_date date
          )
     ORGANIZATION EXTERNAL
          (
           TYPE ORACLE_LOADER
           DEFAULT DIRECTORY employee_data
           ACCESS PARAMETERS
          (
           RECORDS DELIMITED BY NEWLINE
           FIELDS TERMINATED BY ','
           MISSING FIELD VALUES ARE NULL
           )
         LOCATION ('emp.dat')
           )
      REJECT LIMIT UNLIMITED;
Table created.
SQL>
```

There are several important components of this CREATE TABLE statement that you need to be aware of:

- **ORGANIZATION EXTERNAL** Indicates to Oracle that the table you are creating is an external table, not a regular database table.
- **TYPE** Specifies the type of access loader: ORACLE_LOADER or ORACLE_DATA PUMP. The default type is ORACLE_LOADER. However, only the ORACLE_DATA PUMP access loader can perform a data unload. Both access drivers can perform a data load. The `records delimited by newline` clause indicates that each line in the datafile is a new row in the external table. The `fields terminated by ','` clause tells Oracle that each column is seperated by a comma in the datafile. If there are missing values, the clause `missing field values are null` instructs Oracle to treat them as null.
- **DEFAULT DIRECTORY** Allows you to specify a file system as the directory, by first using the CREATE DEFAULT DIRECTORY AS statement.
- **ACCESS PARAMETERS** Describes the *structure of the external data*. The access parameters ensure that the data from the data source is processed correctly to match the definition of the external table.
- **LOCATION** Refers to the actual dump file location. You must specify a dump filename at least. In addition, you may specify an optional directory name as well. If you furnish just a dump filename and no directory name, Oracle will automatically place the dump file in the default dump directory. Note that both of the following location specifications are valid:

  ```
  LOCATION('dept_xt.dmp')
  LOCATION(dept_xt_dir.dep_xt.dmp)
  ```

- **REJECT LIMIT UNLIMITED** Specifies that there is no limit on the number of errors that occur during the querying of the external tables.

Loading and Unloading Data

The terms *loading* and *unloading* in the context of external tables can be confusing, so let's pause and make sure you undertand these terms without any ambiguity. When you deal with external tables, this is what these terms mean:

- *Loading* data means reading data from an external table and loading it into a regular Oracle table. Oracle first reads the data stream from the files you

specify. Oracle will then convert the data from its external representation to an Oracle internal datatype and pass it along to the external table interface.

- *Unloading* data means reading data from a regular Oracle table and putting it into an external table. You couldn't do this in the Oracle9*i* database.

As I explained earlier, only the `ORACLE_DATA PUMP` access driver can perform an external table population (unloading data). Why is the new functionality (unloading data into external tables) important? Following are some of the benefits of this new Oracle Database 10*g* feature:

- Loading table data into flat files means that you can now store data or move it to different databases easily. If you want to move large volumes of data across platforms, external tables provide a means of doing so, since the external files are platform-independent.
- During the population of data warehouses, there are many situations where you need to perform complex ETL jobs. You can use SQL transformations to manipulate the data in the external tables before reloading them into the same or other databases.
- Once you create an external table and populate it using the CTAS statement, you can move the text files containing data and create new external tables in the same or a different database.

Note that when you talk about to writing to external tables, you are really referring to writing to an external file. You use a `SELECT` statement to extract table data to this operating sytem file. The `ORACLE_DATA PUMP` access driver writes data to this file in a binary Oracle-internal Data Pump format. You can then use this file to load another external table on a different database.

The following example shows how you can create an external table and populate it with data from an external flat file. The only difference between this example and the preceding external table creation statement is that it uses the `ORACLE_DATAPUMP` access driver rather than the `ORACLE_LOADER` driver.

```
SQL> CREATE TABLE inventories_xt2
  2  (
  3    product_id          NUMBER(6),
  4    warehouse_id        NUMBER(3),
  5    quantity_on_hand    NUMBER(8)
  6  )
  7  ORGANIZATION EXTERNAL
```

```
  8  (
  9    TYPE ORACLE_DATA PUMP
 10    DEFAULT DIRECTORY def_dir1
 11    LOCATION ('inv_xt.dmp')
 12  );
Table created.
SQL>
```

The CREATE TABLE ... ORGANIZATION EXTERNAL statement creates an external table. There is no data in this table at this point. The external table inventories_xt2 is then populated using the flat file inv_xt.dmp, located in the directory def_dir1. You could do all this in Oracle9*i*. The feature shown in the next example—writing to an external table—is a brand-new Oracle Database 10*g* external tables enhancement.

```
SQL> CREATE TABLE dept_xt
  2       ORGANIZATION EXTERNAL
  3     (
  4       TYPE ORACLE_DATA PUMP
  5       DEFAULT DIRECTORY ext_tab_dir1
  6       LOCATION ('dept_xt.dmp')
  7      )
  8*      AS SELECT * FROM scott.DEPT
SQL> /
Table created.
```

exam Watch

Remember that when you load an Oracle table from an external table (data loading), you use the INSERT INTO ...SELECT clause. When you populate an external table using Oracle table data, you use the CREATE TABLE AS SELECT clause.

If you now go look in the location specified for the default directory (ext_tab_dir1), you'll see a dump file named dept_xt.dmp, which contains the data from the DEPT table. You can then use this dump file in the same database or a different database to load other tables. Note that you must create the default directory ext_tab_dir1 beforehand for this external table creation statement to succeed. You are creating the external table dept_xt as an external table. The table structure and data both come from the regular Oracle DEPT table. The CTAS method of table creation will load the data from the DEPT table into the new external table dept_xt.

Where will the data for the dept_xt be located? You'll be wrong if you answer something like "in a table segment"! Since the dept_xt table is defined as an external table, the CTAS command simply stores the table data in the external file called dept_xt_dmp. Thus, the external table is really composed of proprietary format, operating system-independent flat files.

exam Watch

When you use the external tables feature to extract table data to a file, you export only the data. You can't export metadata using external tables. If you wish to extract the metadata for any object, just use DBMS_METADATA, as shown here:

```
SET LONG 2000
SELECT DBMS_METADATA.GET_DDL('TABLE','EXTRACT_CUST') FROM DUAL;
```

Parallel Population of External Tables

Since external tables can be quite frequently very large, it's nice to know that you can load external tables in a parallel fashion, simply by using the keyword `PARALLEL` when creating the external table. Here's the catch with the `PARALLEL` command while creating an external table: your use of the `PARALLEL` parameter will make sense only if you have more than one file specified as values for the `LOCATION` variable. Otherwise, the `PARALLEL` command really can't do anything in parallel (the degree of parallelism defaults to 1)! The reason for this is that Oracle will allocate exactly only one parallel execution server for each file. If you specify `PARALLEL=4` and specify two datafiles for Oracle to write to, your degree of parallelism is automatically lowered to 2. Thus, the degree of parallelism is constrained by the number of dump files you specify under the `LOCATION` parameter.

Here's an example of how to use the `PARALLEL` command while creating external tables:

```
SQL> CREATE TABLE inventories_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATA PUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_xt.dmp1','inv_xt.dmp2',inv_xt.dmp3')
     )
  7  PARALLEL
  8  AS SELECT * FROM inventories;
```

CERTIFICATION OBJECTIVE 2.05

Defining External Table Properties

The data dictionary view DBA_EXTERNAL_TABLES describes features of all external tables in your database:

```
SQL> desc dba_external_tables
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------
 OWNER                                     NOT NULL VARCHAR2(30)
 TABLE_NAME                                NOT NULL VARCHAR2(30)
 TYPE_OWNER                                         CHAR(3)
 TYPE_NAME                                 NOT NULL VARCHAR2(30)
 DEFAULT_DIRECTORY_OWNER                            CHAR(3)
 DEFAULT_DIRECTORY_NAME                    NOT NULL VARCHAR2(30)
 REJECT_LIMIT                                       VARCHAR2(40)
 ACCESS_TYPE                                        VARCHAR2(7)
 ACCESS_PARAMETERS                                  VARCHAR2(4000)
 PROPERTY                                           VARCHAR2(10)
```

Pay particular attention the last three columns in the DBA_EXTERNAL_TABLES view:

- The ACCESS_TYPE column refers to whether you have BLOB or CLOB type of access parameters for your external table.
- The ACCESS_PARAMETERS column shows all the access parameters you used in creating the external table.
- The PROPERTY column, which refers to the property of the projected columns, could take two values: REFERENCED or ALL.

The default value for the PROPERTY column for all external tables is ALL. This tells the access driver to always process *all the columns* of an external table, not just some. This means that, regardless of which columns you select in a query, the access driver will process all column values. The access driver will validate only those columns without data errors. The access driver will also eliminate any rows that have erroneous column values, even if those columns aren't a part of the SELECT query.

If the PROPERTY column shows the value REFERENCED, this means that only those columns referenced by a SQL statement are processed (parsed and converted) by the Oracle access driver. When would you want to specify the REFERENCED

property? You do this when you are quite sure about the quality of your data fields and expect no data rejections due to data format errors. For example, you may have a column called emp_id, which you define as a `number(5)` column. When Oracle encounters a row in the datafile where the emp_id has six digits, it would normally reject this row, since the default value for the property is ALL. This means that even if you issue a query that selects a different column, say social_security_num from the employee table, Oracle will reject all rows that have bad data in the emp_id column.

How do you change the default ALL property value for an external table to REFERENCED? Say that you want to change this property for your external table dept_xt, which now has the default ALL property.

```
SQL> select table_name,property from dba_external_tables;
TABLE_NAME                     PROPERTY
------------------------------ ----------
DEPT_XT                        ALL
```

To do this, you use the `ALTER TABLE` command in the following manner.

```
SQL> alter table dept_xt
  2  project column referenced;
Table altered.
SQL> select table_name,property from dba_external_tables;
TABLE_NAME                     PROPERTY
------------------------------ ----------
DEPT_XT                        REFERENCED
SQL>
```

Changing the PROPERTY column to REFERENCED is a good idea if you know your data is clean, because it provides better performance, since *only the projected columns* are parsed and converted. If your data is clean, flagging an external table as REFERENCED will provide better performance when you query only a subset of the columns.The default ALL property projects all columns and will guarantee consistent results, but all of the data is queried for every type of query, thus hindering performance.

CERTIFICATION OBJECTIVE 2.06

Transporting Tablespaces Across Different Platforms

Suppose you need to move a large amount of data between two databases. What's the fastest way to do this? You can use the Data Pump export and import utilities, of

course, to perform the job. However, there is a much faster way to perform the data transfer: use *transportable tablespaces*. Transportable tablespaces simply involve moving tablespaces from one database to another. All you really need to do is to copy the datafiles (containing tables, indexes, and other Oracle database objects) that comprise the tablespace, from the target to the source server (if the two databases reside on different servers) and just import the metadata of the objects in the tablespace to the target database.

Transportable tablespaces are ideal for moving large amounts of data quickly between two databases. The transportable tablespaces feature, of course, has been available for a while now, but both the source and target databases needed belong to the same operating system platform. In Oracle Database 10*g*, for the first time, you can transport tablespaces between *different platforms*. Now the onerous requirement of identical operating system platforms is gone, and you can easily transport a tablespace from pretty much any platform to any other. This is a great feature, as it enables you to take tablespaces from a data warehouse and plug them into data marts, even though your data warehouse runs on an UNIX platform and the data marts are located on smaller Windows servers. As you have probably already figured out, the key here is Oracle Database 10*g*'s ability to convert one set of datafiles from one operating system format to another, so the target database can read the source database files.

Transportable tablespaces are a good way to migrate a database between different platforms.

Transporting a Tablespace Between Identical Platforms

Although the procedure of transporting tablespaces hsn't really changed in Oracle Database 10*g*, let's recap the steps involved in transporting tablespaces, so you can understand the new changes better. Transporting a tablespace from one database to another *when both databases belong to the same platform* consists of the following steps:

1. Ensure that the tablespaces are self-contained.
2. Make the tablespaces read-only.
3. Export the metadata using Data Pump export.
4. Copy the datafiles over to the target system.
5. Use Data Pump import to import the metadata.

Following are the general requriements for transporting tablespaces between two databases:

- Both platforms should use the same character sets.
- Both databases must be using Oracle8*i* or a higher version, but the database version does not need to be identical.
- You cannot transport the SYSTEM tablespaces or any objects owned by the user SYS.
- If you want to transport a partitioned table, all the partitions must be included in the transportable table set. If you are transporting indexes, you also need to transport the tablespaces containing the respective tables as well.
- You can transport tablespaces to a target database only if it has the same or higher compatiblity setting than the source database.

exam Watch

If either the source or the target database compatibility level is less than 10.0.0, you cannot transform a tablespace across different operating sytem platforms.

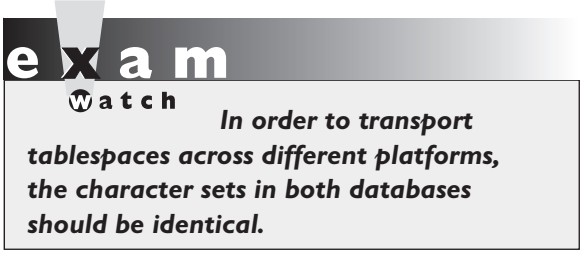
Determining the Supported Platforms

Note that you can't transport tablespaces between all operating system platforms automatically, even in Oracle Database 10*g*. How do you know which platforms are supported for cross-platform tablespace transport? All you need to do to get this information is to query the new `V$TRANSPORTABLE_PLATFORM` view:

```
SQL> col platform_name format a30
SQL> select * from v$transportable_platform;
PLATFORM_ID PLATFORM_NAME                  ENDIAN FORMAT
----------- ------------------------------ --------------
          1 Solaris[tm] OE (32-bit)        Big
          2 Solaris[tm] OE (64-bit)        Big
          7 Microsoft Windows IA (32-bit)  Little
         10 Linux IA (32-bit)              Little
          6 AIX-Based Systems (64-bit)     Big
          3 HP-UX (64-bit)                 Big
          5 HP Tru64 UNIX                  Little
```

```
             4 HP-UX IA (64-bit)              Big
            11 Linux IA (64-bit)              Little
            15 HP Open VMS                    Little
             8 Microsoft Windows IA (64-bit)  Little
             9 IBM zSeries Based Linux        Big
            13 Linux 64-bit for AMD           Little
            16 Apple Mac OS                   Big
            12 Microsoft Windows 64-bit for A Little
               MD
15 rows selected.
SQL>
```

The V$TRANSPORTABLE_PLATFORM view shows all platforms supported for transporting tablespaces. The PLATFORM_NAME column shows all the platforms that are eligible for transporting across platforms. If both your source and target platforms are in this list, you can conduct the transportable tablespaces operation between the databases running on those platforms.



You can find out your own platform name, in case you aren't sure, by running the following simple query:

```
SQL> select platform_name from v$database;
PLATFORM_NAME
----------------------------------------
Microsoft Windows IA (32-bit)
SQL>
```

If you need to transport read-only tablespaces to ensure that your datafile headers can identify the operating system platform, you must first make the datafile read/write at least once (after setting the database compatibility level at 10.0.0 or higher). If your source database is operating at a 9.2.0 compatibility level, for example, you need to first advance the compatibility level to 10.0.0 before you can transport any tablespaces for this database.

Converting to Match Datafile Endian Formats

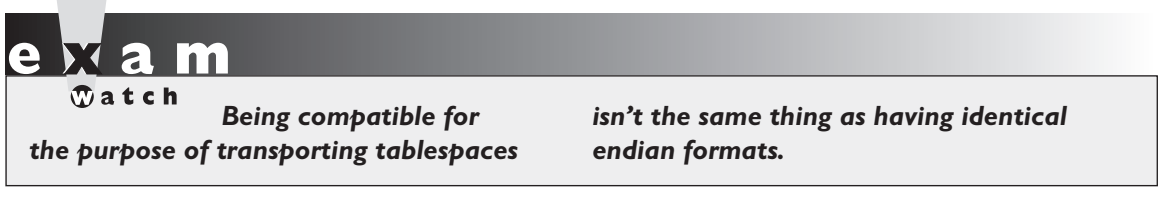
Even if your source and target operating system platforms are identical, you may still not be able to perform a tablespace transport directly. Remember that the most time-consuming job during a tablespace transport is the copying of the files that belong to the tablespace. If the *endian format* of two operating system platforms is different, you need to perform a conversion of the datafiles, either before or after you copy the files to the target system.

What does the ENDIAN FORMAT column, which you can see in the following query, stand for?

```
SQL> select * from v$transportable_platform;
PLATFORM_ID PLATFORM_NAME                   ENDIAN FORMAT
----------- ------------------------------ --------------------------
          1 Solaris[tm] OE (32-bit)                          Big
          2 Solaris[tm] OE (64-bit)                         Big
          7 Microsoft Windows IA (32-bit)                   Little
         10 Linux IA (32-bit)                               Little
SQL>
```

Endian format refers to *byte ordering* in the datafiles of a given platform. Byte ordering affects the way data is written and read in different platforms. There are only two types of endian formats: little or big. In order for the datafiles between two compatible platforms to be transported directly from one another, their endian format (also known as *endianness*) should be the same. Both the source and target platforms should have an identical endian format—either both are in the *big* format or both are in the *little* format.

If you have two platforms that are in the compatible list for transporting tablespaces, but their endian formats are different, you need to convert the datafiles belonging to the tablespaces that you are exporting, using the RMAN utility.



Transporting a Tablespace Across Platforms

The steps for transporting tablespaces across platforms are the same as for transporting tablespaces across identical platforms, with an additional step if the source and target database file endian format are different. The following sections provide an example of these steps.

Ensure the Tablespaces Are Self-Contained

Ensure that the tables you want to transport all are placed in their own separate tablespaces. To ensure that your tablespaces are self-contained, you need to use the `TRANSPORT_SET_PROCEDURE` in the Oracle-supplied package `DBMS_TTS`.

Make the Tablespaces Read-Only

Alter the tablespace to make it read-only. Once you complete the export of the metadata in the next step, you can make the tablespace read/write again.

Export the Metadata Using Data Pump Export

Export the metadata describing the objects in the tablespace(s), by using the `TRANSPORTABLE_TABLESPACES` parameter.

Convert the Datafiles to Match Endian Format

If your platforms are compatible, but the endian formats are different, you need to convert the datafiles. You may perform the conversion before transporting the tablespace set or after finishing the transport. You can convert the datafiles before transporting the tablespaces, using the following `CONVERT TABLESPACE` command in the RMAN:

```
RMAN> convert tablespace finance_tbs01
2> to platform 'HP-UX (64-bit)'
3> format '/temp/%U';
Starting backup at 09-MAY-04
using channel ORA_DISK_1
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00011 name=C:\ORACLE\TEST02.DBF
converted datafile=C:\TEMP\DATA_D-FINANCE_I-2343065311_TS-TODAY_FNO-11_05FLAUM6
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:17
```

```
Finished backup at 09-MAY-04
RMAN> exit
Recovery Manager complete.
```

In this example, I show how you can use the FORMAT parameter to tell Oracle what format the newly converted file should be and in which directory to put it. But as you can see, Oracle gives the file a name. If you want to specify the datafile name youself, perform the conversion using the DB_FILE_NAME_CONVERT clause, as follows. (Remember that you use the following command when you convert the files directly on the source system, before transporting them.)

```
RMAN> convert tablespace today
2> to platform 'HP-UX (64-bit)'
3> db_file_name_convert = 'c:\oracle\test02.dbf','c:\temp\test02.dbf';
Starting backup at 10-MAY-04
using target database controlfile instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=151 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00011 name=C:\ORACLE\TEST02.DBF
converted datafile=C:\TEMP\TEST02.DBF
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:16
Finished backup at 10-MAY-04
RMAN>
```

The DB_FILE_NAME_CONVERT clause performs the following functions for you:

- Takes a given filename and converts it to any filename you specify
- Places the converted file in the location you specify

Copy the Files to the Target System

At this point, you need to copy both the *converted* datafile that is part of the tablespace (finance_tbs01 in this example) as well as the expdp dump file, sales2003.dmp, over to the target server where your target database is running.

If you chose to transport the tablespaces (the datafiles that constitute the tablespaces) first, you must convert the datafiles on the target platform at this point, before trying to perform the import of the metadata in the tablespace. Here's an example that

shows how you can take a datafile that belongs to the HP-UX operating system platform and convert it into a Windows platform:

```
RMAN> convert datafile
2>    'c:\audit_d01_01.dbf'
3>    to platform 'Microsoft Windows IA (32-bit)'
4>    from platform='HP-UX (64-bit)'
5>    FORMAT '\export';
```

As in the previous case where you performed the file conversion on the source sytem, you may use the `DB_FILE_NAME_CONVERT` clause when performing the data file conversion on the target system. Your datafile conversion statement would then have the format `CONVERT DATAFILE ...FROM PLATFORM ...DB_FILE_NAME_CONVERT = ....`

By default, Oracle places the converted files in the Flash Recovery Area, without changing the datafile names.

Use Data Pump Import to Import the Metadata

Once you move the converted files files over to the target system (or move the files over first and convert them later), use the Data Pump import utility as follows to import the metadata into the target database—that is, just plug in the tablespaces and use the Data Pump import to integrate the datafiles and their metadata (found in the test.dmp file):

```
impdp system/manager TRANSPORT_TABLESPACE=y
DATAFILES='/u01/app/oracle/data01.dbf'
TABLESPACES=test FILE=test.dmp
```

Understanding Exceptions to File Conversions

There is an important exception to the file conversions that the RMAN performs for you in order to enable the transporting of tablespaces between two platforms with different endian formats. RMAN doesn't automatically convert CLOB data for you during file conversion. Here's what you need to know about CLOBs and file conversion for transportable tablespaces:

- If your CLOBs were created in an operating system with the big-endian format, you don't need to convert the CLOB data.
- If you are transporting little-endian CLOB data to a big-endian system, a conversion must be done. But even here, Oracle can automatically convert

the CLOB data to the big-endian format dynamically, whenever a user accesses the data. However, if you wish to do all the conversion yourself, you may do so by re-creating the table on the target database.

Here's the reasoning for the variation in the treatment of CLOB data in big-endian and little-endian systems: prior to Oracle Database 10*g*, CLOBs were stored in the UCS2 format, which is endian-dependent, but Oracle Database 10*g* stores them as AL16UTF16, which is endian-independent. However, big-endian UCS2 is the same as AL16UTF16. So if your source system was big-endian, there isn't any difference. If it wasn't, Oracle will convert the datafiles on demand.

INSIDE THE EXAM

The exam will test your knowledge of the new Data Pump import and export parameters. You must understand the new parameters like the remapping parameters of Data Pump import. What are the different levels at which you can perform transformations during an import? Pay particular attention to the new parameters like `ESTIMATE`, `ESTIMATE_ONLY`, `NETWORK_LINK`, `INCLUDE`, and `EXCLUDE`. You must know the levels at which you can perform Data Pump import and export.

You can expect questions on the interactive import and export commands. How do you attach to a running job? What happens to a job and a client session when you stop a job? Review the Data Pump dictionary views like `DBA_DATAPUMP_JOB_STATISTICS` and `DBA_DATAPUMP_JOBS`.

There will be a question on the Data Pump architecture. You must know the difference between the External Table API and Direct Path API. You can expect some questions on the `DBMS_DATAPUMP` and the `DBMS_METADATA` packages. You must understand the importance of the master table.

There will be questions on the external table population feature. You must remember the syntax for performing an external table population (`CREATE TABLE AS SELECT`). Review the parallel feature, putting special emphasis on the relationship between the degree of parallelism and the number of dump files.

The test will contain questions on the new features related to transportable tablespaces. What is an endian format, and what role does it play in transporting tablespaces across platforms? You must remember the RMAN commands to convert the endian formats of files. What is the difference in the file conversion commands when you convert files on the target platform and when you convert them on the source platform?

CERTIFICATION SUMMARY

This chapter introduced you to the new Data Pump technology in Oracle Database 10*g*. You saw how Data Pump offers sophisticated new features to run large export and import jobs faster, while offering exceptional data-filtering techniques. You also learned about the powerful interactive Data Pump features. You learned how to monitor your Data Pump jobs.

You learned how to populate external tables. You also saw how the new options you have in Oracle Database10*g* regarding external table properties. You learned how to transport tablespaces across server platforms, even when the file semantics are different between two operating sytem platforms.

TWO-MINUTE DRILL

Introduction to the Data Pump Architecture

- ❑ The new Data Pump technology is a much more efficient way of moving large amounts of data than the old export/import utilities.
- ❑ You can still continue to use the traditional export and import utilties in Oracle Database 10*g*.
- ❑ Data Pump technology is entirely server-based; all work takes place on the server.
- ❑ The Data Pump export utility exports data out of the database, and the Data Pump import utility imports data into a database.
- ❑ There are two kinds of parameters: a set of command-line parameters and a special set of interactive commands in both the Data Pump import and export utilities.
- ❑ The Oracle-supplied package `DBMS_DATA PUMP` is used to implement the Data Pump API.
- ❑ The clients for the Data Pump export and import utilities are `expdp` and `impdp`, respectively.
- ❑ `DBMS_METADATA`, an older Oracle-provided package, is used to extract and modify data dictionary metadata.
- ❑ Data Pump export can access data in two ways: direct-path access using the Direct Path API or through external tables. Data Pump itself makes the decision as to the access choice, based on which will be faster in a given case.
- ❑ Direct-path access is the first method Oracle will try to use. Under some conditions, Oracle cannot use the direct-path method, and it must use the external tables access method.
- ❑ You can export data using either direct path or external tables and import the data back with either of the two methods.
- ❑ There are three types of Data Pump files: dump files, log files, and SQL files.
- ❑ Data Pump export dump files are created on the server, using directory objects.
- ❑ Directory objects are named objects that are mapped to an operating system directory.
- ❑ Once you create a directory, you can access that file system by simply referring to the dirctory name.

- In order to create a directory, you must have the DBA role. In order to use a directory, you must have the appropriate read and/or write privileges on the directory.
- DATA_PUMP_DIR is the default directory object for Data Pump jobs. Only privileged users can use this default directory object.
- The order of precedence for file locations is the following: the directory name as part of a file parameter name, the value assigned to the DIRECTORY parameter, the directory name specified by DATA_PUMP_DIR environemnt variable, and finally, the default value for the DATA_PUMP_DIR object.
- All Data Pump jobs consist of a master and several worker processes. The master process controls jobs and the worker processes as well. The master process is also responsible for monitoring the progress of jobs.
- The master process uses the master table to store the database object location. The master table is created during the export process. If the job completes successfully, the master table is automatically deleted from the database.
- During Data Pump import, the master table is consulted to verify the correct sequencing of objects during import.
- If you choose the PARALLEL option, the worker processes become parallel execution coordinators.
- The benefits of the Data Pump technology include the ability to restart jobs, parallel execution capabilities, ability to attach to a running job, ability to estimate space requirements, fine-grained export and import capabilities, and remapping capabilities.
- You can perform both network mode exports and imports.
- You can perform Data Pump export/import from the command line or with the help of parameter files.
- In Data Pump export/import, you use the interactive mode to intervene during a running job. There are several special commands you can use in this interactive mode.
- You can start the interactive mode either by using the CONTROL-C combination or by using the ATTACH command from a different session.

Using Data Pump Export and Import

- You can perform Data Pump export and import in full, tablespace, table, or schema modes. You can also perform Data Pump jobs in the transportable tablespaces mode.

- ❑ The CONTENT parameter can take the values ALL, DATA_ONLY, or METADATA_ONLY. The ALL parameter enables the export of both data and metadata. The DATA_ONLY parameter lets you export data only. The METADATA_ONLY option enables the export of only the object definitions.
- ❑ The EXCLUDE parameter forces the exclusion of specific objects, and the INCLUDE parameter requries the inclusion of specified objects.
- ❑ The EXCLUDE and INCLUDE parameters are mutually exclusive.
- ❑ You use the QUERY parameter to filter table row data with the help of a SQL statement.
- ❑ The ESTIMATE parameter provides an estimate of the size of the export job. It uses BLOCKS by default. You can specify ESTIMATE=STATISTICS to make the parameter use database statistics instead of the default blocks method.
- ❑ The ESTIMATE_ONLY parameter just gives you a size estimate, without performing an export.
- ❑ You can connect to a running export or import job by using the ATTACH command.
- ❑ The CONTINUE_CLIENT parameter takes you out of the interactive mode but keeps the job running. The EXIT_CLIENT command will terminate the interactive session and the client session. The KILL_JOB command will terminate the export or import job in addition. The STOP_JOB command stops running Data Pump jobs.
- ❑ The STATUS parameter will provide you with periodic job progress updates.
- ❑ The default value of the PARALLEL parameter is 1. In practice, it is limited by the number of dump files you provide for the export job.
- ❑ By default, a Data Pump export job will export the entire schema of the user running it.
- ❑ The SQLFILE parameter is used during a Data Pump Import to extract DDL to a specified file, without conducting an import of the data in the export dump file.
- ❑ If you specify REUSE_DATAFILES=Y, Data Pump will overwrite your existing datafiles.
- ❑ Remapping parameters are used during a Data Pump import job to remap database objects. You can remap datafiles, tablespaces, and entire schemas.

- The NETWORK_LINK parameter enables you to import data directly from as target database, without using any dump files. You must first create a database link before performing a network import.
- The TRANSFORM parameter enables you to modify storage and tablespace clauses during an import.

Monitoring a Data Pump Job

- You can monitor Data Pump jobs with the help of the views DBA_DATA PUMP_JOBS, DBA_DATA PUMP_SESSIONS, and V$SESSION_ LONGOPS.
- The DBA_DATAPUMP_JOBS view shows all active Data Pump jobs.
- The DBA_DATAPUMP_SESSIONS view shows all the user sessions attached to an import or export job.
- The V$SESSION_LONGOPS view tells you how far a Data Pump job has progressed.

Creating External Tables for Data Population

- You can now populate external tables by using the ORACLE_DATA PUMP access loader.
- The main parameters you need to specify in the creation of external tables are type, default_directory, location, and access_parameters.
- *Loading* data refers to reading data from external tables. *Unloading* data refers to populating external tables.
- You use the CREATE TABLE AS SELECT (CTAS) statement to populate external tables.
- The ORACLE_LOADER access driver permits only the loading of external tables.
- The ORACLE_DATAPUMP access loader will permit both the loading and the unloading of data (reading as well as writing to external tables).
- You can load external table creation faster by using the PARALLEL parameter. If you use the PARALLEL parameter, you must specify more than one datafile for writing the data. The degree of parallelism is limited by the number of datafiles you provide.

Defining External Table Properties

- ❑ External tables could have either of two values for the PROPERTY column: `ALL` or `REFERENCED`.
- ❑ The default value fo the PROPERTY column is `ALL`.
- ❑ If your data is known to be clean (no data formatting errors), you should use the `REFERENCED` value for the PROPERTY column, for better performance.
- ❑ You can use the `ALTER TABLE` command to change the PROPERTY column of an external table.

Transporting Tablespaces Across Different Platforms

- ❑ You can now transport tablespaces across different operating system platforms.
- ❑ In order to qualify for a cross-platform tablespace transport, both platforms should set to the compatibility equal to 10.0.0, use an identical character set, and be in the compatible platforms list.
- ❑ The view `V$TRANSPORTABLE_PLATFORM` will let you know if a pair of operating system platforms are compatible.
- ❑ The endian format of an operating system platform refers to the byte-ordering format of the files on that platform.
- ❑ If the endian format of two compatible platforms is the same, you don't need to convert the datafiles for transporting them across different platforms.
- ❑ If the endian format of two compatible platforms is different, you must convert the datafiles either before or after you physically transport the tablespaces.
- ❑ You use the `DB_FILE_NAME_CONVERT` option to convert file formats from one endian format to another.
- ❑ If your CLOBs were created in an operating sytem with the big-endian format, you don't need to convert the CLOB data.
- ❑ If you are transporting little-endian CLOB data to a big-endian system, you must convert the data.

SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. Read all the choices carefully because there might be more than one correct answer. Choose all correct answers for each question.

Introduction to the Data Pump Architecture

1. Which of the following are part of the new Data Pump technology?
   A. `DBMS_METADATA`, `DBMS_DATA PUMP`, Direct Path API
   B. `DBMS_METADATA`, `DBMS_DATA PUMP`, Indirect Path API
   C. `DBMS_METADATA`, `DBMS_DATA PUMP`, SQL Loader API
   D. `DBMS_METADATA`, `DBMS_DATA PUMP`, Export API
2. What is the `DBMS_METADATA` package used for?
   A. Transport tablespaces between two databases
   B. Load and unload metadata
   C. Perform a cross-platform transport of tablespaces
   D. Load external tables
3. Assume the following is the first import job you are performing in your database:

```
$ impdp system/manager parfile=imp.par
```

   What would be the default name of your master table?
   A. IMPORT_FULL_01
   B. SYS_IMPORT_FULL_01
   C. SYSTEM_IMPORT_FULL_01
   D. DATA PUMP_JOB_FULL_01
4. Which of the following statements is correct?
   A. The master table is created during a Data Pump export job and written to the dump file at the very beginning of the export job.
   B. The master table is created during a Data Pump export job and written to the dump file at the very end of the export job.
   C. The master table is created during a Data Pump import job and written to the dump file at the very beginning of the import job.
   D. The master table is created during a Data Pump import job and written to the dump file at the very end of the import job.

Using Data Pump Export and Import

5. The DBA has just performed a full database Data Pump export. She must now perform a full database import using the dump file set created by the export job. Which one of the following statements would be correct under the circumstances?
   A. `impdp system/manager dumpfile=expdata.dmp FULL=Y`
   B. `impdp system/manager dumpfile=expdata.dmp`
   C. `impdp system/manager FULL=Y`
   D. `impdp system/manager dumpfile=expdata.dmp FROMUSER=TOUSER`
6. Which of the following parameters enables you to perform a data-only unloading of data?
   A. `EXCLUDE`
   B. `INCLUDE`
   C. `CONTENT`
   D. `DATA_ONLY`
7. Which of the following statements is correct?
   A. If you stop a job using the `STOP_JOB` command, the master table is retained for use in restarting the job.
   B. If you stop a job using the `KILL_JOB` command, the master table is dropped and the job cannot be restarted.
   C. If you stop a job using the `KILL_JOB` command, the master table is retained and you can restart the job later.
   D. If a job terminates unexpectedly, the master table is dropped automatically.
8. Which of the following occurs when you start an interactive session with Data Pump?
   A. The currently running export job is interrupted briefly.
   B. The current job continues normally.
   C. The current job is stopped and you need to restart it later.
   D. You cannot start an interactive session when a Data Pump job is running.

Monitor a Data Pump Job

9. How can you see the amount of work performed so far by your Data Pump import job?
   A. Query the `V$JOB_STATUS` view
   B. Query the `V$SOFAR` view
   C. Query the `V$SESSION_LONGOPS` view
   D. Query the `DBA_DATAPUMP_JOBS` view

10. Which is the sequence of commands you must use if you want to suspend and then resume your Data Pump job?

A. `START_JOB, ATTACH, CONTINUE_CLIENT`
B. `ATTACH, KILL_SESSION, CONTINUE_CLIENT`
C. `ATTACH, STOP_JOB, START_JOB`
D. `STOP_JOB, ATTACH, CONTINUE_CLIENT`

11. What information do you need to supply to attach to a running job from a different session?

A. The location of the dump file
B. Username and password, job name
C. Username and password onl
D. Username and password, master table name

12. What does the column SOFAR in the monitoring view `V$SESSION_LONGOPS` tell you?

A. How many megabytes have been transferred thus far in the job
B. What percentage of the job has been completed thus far
C. What percentage of the job remains to be done
D. The estimated number of megabytes left to be transferred

Creating External Tables for Data Population

13. Which *two* of the following statements is true?

A. You use the `ORGANIZATION EXTERNAL` clause during an external table unloading operation.
B. You use a `CREATE TABLE AS SELECT` statement during an external table loading operation.
C. You use a `CREATE TABLE AS SELECT from` statement during an external table unloading operation.
D. You use the `ORGANIZATION EXTERNAL` clause only for reading data into an external table from an operating system file.

14. What does unloading of data refer to?

A. The reading of data from external datafiles into external tables.
B. The writing of data from Oracle tables to external datafiles.
C. The writing of data from external datafiles into external tables.
D. The reading of data from Oracle tables into external datafiles.

15. Why should you use the `NETWORK_LINK` export parameter when you perform a Data Pump export of a read-only database?

A. You can't write to a read-only database.
B. The export is faster if you use the `NETWORK_LINK` parameter.
C. You don't need to use export dump files when you use the `NETWORK_LINK` parameter during export
D. You can't use the traditional export utility to export a read-only database.

16. If the number of files in the `LOCATION` clause is different from the degree of parallelism that you specify, which two statements below would be correct?

A. Oracle will ignore the `PARALLEL` parameter.
B. Oracle will perform the table population with the same degree of parallelism as the number of files.
C. Oracle ignores any extra files (files greater than the degree of parallelism) that you may specify.
D. You can instruct the parallel execution server to write to multiple files simultaneously.

Defining External Table Properties

17. The DBA knows that the database may reject certain columns in an external table, due to data format errors. In this case, what should the DBA do to get consistent query results?

A. Clean up the data so the rows with data format errors are taken out of the table.
B. Alter the external table to set the PROJECT COLUMN attribute to ANY.
C. Alter the external table to set the PROJECT COLUMN attribute to ALL.
D. Alter the external table to set the PROJECT COLUMN attribute to REFERENCED.

18. Which of the following is true in Oracle Database 10*g*?

A. The default value for an external table PROJECT COLUMN attribute projects all columns.
B. The default value for an external table PROJECT COLUMN attribute projects no columns.
C. The default value for an external table PROJECT COLUMN attribute projects only columns with no data errors.
D. The default value for an external table PROJECT COLUMN attribute projects only columns with data errors.

19. Which of the following is true if your data is clean (without any formatting errors)?

A. Using the `ALL` value for the PROJECT COLUMN attribute always provides the same results.

B. Using the `REFERENCED` value for the PROJECT COLUMN attribute always provides the same results.

C. Using the default value for the PROJECT COLUMN attribute always provides the same results.

D. Using the `ALL` value for the PROJECT COLUMN attribute always provides different results.

20. Of the following, which statement is true of the external table properties?

A. `REFERENCED` is better because you need to parse and convert only some columns.

B. `ALL` is better because you need to parse and convert only some columns.

C. `ALL` is better because you need to parse and convert all columns.

D. `REFERENCED` is better because you need to parse and convert all columns.

Transporting Tablespaces Across Different Platforms

21. Which of the following interfaces can you use to convert your database files when the endian formats are incompatible between a source and a target database?

A. SQL*Plus

B. RMAN

C. OEM Database Control

D. Oracle PL/SQL procedures and packages

22. Which of the following do you need to do when the endian formats of the target and source database files are different?

A. Convert the source files on the source system, copy them to the target system, and import the metadata.

B. Convert the source files on the source system, copy them to the target system, and export the metadata.

C. Copy the source files to the target system, convert the datafiles, and export the metadata.

D. Copy the source files to the target system, convert the datafiles, and import the metadata.

23. To find out if two databases are cross-transportable compliant, which data dictionary view do you need to use?

A. `V$TRANSPORTABLE_PLATFORM`
B. `V$ENDIAN FORMAT`
C. `V$PLATFORM`
D. `V$COMPATIBILITY_LEVEL`

24. Which of the following can you do if you find both the target and source operating system platforms in the `V$TRANSPORTABLE_TABLESPACES` view?

A. Automatically transport tablespaces between the two platforms
B. Transport tablespaces only after you perform a mandatory file conversion first
C. Transport tablespaces between the two platforms only if their endian format is different
D. Transport tablespaces between the two platforms, provided you always perform a file conversion first if the file endian formats are different

LAB QUESTION

Start a Data Pump export job as the user SYSTEM. Export the entire database. Show the commands you would enter to perform the following actions:

- Start an interactive session by using the `ATTACH` command.
- Find out the name of the master table.
- Parallelize the export (four streams).
- Resume the export job.

SELF TEST ANSWERS

Introduction to the Data Pump Architecture

1. ☑ **A.** The `DBMS_METADATA` and `DBMS_DATAPUMP` packages are the main Oracle PL/SQL packages that the Data Pump technology uses. The Direct Path API is a part of Data Pump as well.
   ☒ **B, C,** and **D** all contain the name of an invalid API.

2. ☑ **B**. As the name of the package indicates, `DBMS_METADATA` is used to load and unload metadata.
   ☒ **A** and **C** are not correct because the package doesn't play a role in transporting tablespaces. **D** is wrong because the package isn't related to external tables.

3. ☑ **C.** The default name of the master table is of the format: *USERNAME_OPERATION_TYPE_N*. In this case, since you know user SYSTEM is performing a full import, it shouldn't be that hard to pick this answer.
   ☒ **A** and **D** are wrong since they don't have either user's name. **B** is wrong since it contains the username SYS instead of SYSTEM.

4. ☑ **B.** The master table is created during an export job. The creation of the master table is the last thing that the Data Pump export utility does, before finishing the job.
   ☒ **A** is wrong since it says that the master table is created at the *beginning* of the export job. **C** and **D** are incorrect since they state that the master table is created during the import job.

Using Data Pump Export and Import

5. ☑ **A**. This answer provides all the necessary parameters: username, dump filename, and the `FULL` parameter to perform the full import.
   ☒ **B** is wrong because it is missing the `FULL` parameter—the default mode for import is the schema level, not a full database import. **C** is wrong because it is missing the dump filename. **D** is incorrect because there isn't a `FROMUSER/TOUSER` option in Data Pump.

6. ☑ **C.** This is a slightly tricky question. The `CONTENT` parameter offers the option of exporting just the data in the tables, by using `DATA_ONLY` as the value for the parameter.
   ☒ **A** and **B** are incorrect since these parameters enable you to specify only the type of objects you want to include or exclude. **D** is wrong since `DATA_ONLY` is not an export parameter—it's an option for the `CONTENT` parameter.

7. ☑ **A** and **B.** A is correct because using the `STOP_JOB` command doesn't drop the master table. **B** is correct because using the `KILL_JOB` command terminates the job and drops the master table.

☒ **C** is incorrect since the use of the `KILL_JOB` command removes the master table. **D** is incorrect since an unexpected termination of a job doesn't automatically drop the master table.

8. ☑ **B.** You can start an interactive session in Data Pump only when an export job is already running. Thus, when you log interactively in to a Data Pump job using either the CONTROL-C sequence or the `ATTACH` command, the job is already running.
☒ **A** and **C** are wrong since the export job is neither interrupted nor stopped when you log in interactively. **D** is wrong since the Data Pump job must be running for you to log in interactively.

Monitor a Data Pump Job

9. ☑ **C.** The SOFAR column in the `V$SESSION_LONGOPS` view tells you how much of Data Pump job (in megabytes) has been completed thus far.
☒ **A, B,** and **D** cannot tell you anything about the progress of your Data Pump jobs.

10. ☑ **C.** First, you need to use the `ATTACH` command to attach to the interactive session. To suspend the job, you should use the `STOP_JOB` command. The `START_JOB` command will resume the job.
☒ **A, B,** and **D** provide various wrong sequences of commands.

11. ☑ **B.** You need to provide both the username/password and the job name before you can attach to a running job.
☒ **A** is wrong because you don't need to specify the dump file location. **C** is wrong because the username/password is inadequate to attach to a session. **D** is wrong because you don't need the master table's name to attach to a Data Pump session.

12. ☑ **A.** The SOFAR column tells you how many megabytes have been transferred thus far.
☒ **B** and **C** are incorrect because the SOFAR column doesn't deal with the percentage of work—it deals with the work in terms of megabytes. **D** is wrong because it talks about work still to be done, not work already completed by the job.

Creating External Tables for Data Population

13. ☑ **A** and **C. A** is correct because you must use the `ORGANIZATION EXTERNAL` clause whether you are loading or unloading data. **C** is correct because you have to use the `CREATE TABLE AS SELECT` (CTAS) clause when you populate an external table.
☒ **B** is incorrect because you don't need to use the CTAS statement during external table loading. **D** is incorrect since you need the `ORGANIZATION EXTERNAL` clause for both reading from and writing to external tables.

14. ☑ **B.** Unloading of data is the writing of Oracle table data to external datafiles, in the form of external tables.
☒ **A** and **D** are clearly wrong since unloading of data involves writing of data, not reading of data. **C** is wrong since it states the opposite of what's true.

15. ☑ **A** is correct because you can't use Data Pump export if you are exporting a read-only database, since you can't create the master table in a read-only database. The `NETWORK_LINK` parameter provides a way to solve this problem.
☒ **B** is wrong because this isn't the reason why you need to use the `NETWORK_LINK` parameter. **C** is incorrect because you do need to create an export dump file, no matter what parameters you specify. **D** is wrong because you can export a read-only database using the traditional export utility.

16. ☑ **B** and **C. B** is correct because Oracle will set the degree of parallelism to the number of files. **C** is correct because when the number of datafiles is more than the degree of parallelism, Oracle will ignore the extra files.
☒ **A** is incorrect since Oracle doesn't ignore the `PARALLEL` parameter if the number of files is different from the degree of parallelism. **B** is incorrect since Oracle doesn't adjust the degree of parallelism to match the number of datafiles. **D** is wrong since the parallel execution server will not write to multiple files simultaneously.

Defining External Table Properties

17. ☑ **C.** The DBA should make sure that the PROJECT COLUMN attribute value is set to ALL, which is the default value for the column as well.
☒ **A** is incorrect since the DBA isn't responsible for cleaning up the data format errors in the data. **B** is wrong since there isn't a value of ANY for the PROJECT COLUMN. **D** is wrong since setting the column value to REFERENCED will give you inconsistent query results, depending on the columns you specify in each query.

18. ☑ **A.** The default value for the PROJECT COLUMN attribute is ALL, which means Oracle will project all columns out.
☒ **B** is wrong since the default behavior, as answer **A** shows, is to project out all columns. **C** and **D** are wrong since the projection of the columns has nothing to do with whether the columns have data errors or not.

19. ☑ **A, B,** and **C.** If your data is clean, it doesn't make a difference if the PROJECT COLUMN attribute has the value ALL or REFERENCED, since the results are going to be consistent with either value.
☒ **D** is incorrect since the use of the ALL column will produce the same results for any query, if your data doesn't have any formatting errors.

20. ☑ **A.** REFERENCED is better because you need to parse and convert only the selected columns in the query, not all the columns in the table.
☒ **B** is incorrect because you need to parse and convert all columns, not just some, if you choose the ALL value for the PROPERTY column. **C** is incorrect since ALL forces the parsing and conversion of all columns; it isn't better than using the REFERENCED value for the PROPERTY column. **D** is incorrect since REFERENCED means that Oracle doesn't parse and convert all columns.

Transporting Tablespaces Across Different Platforms

21. ☑ **B.** You need to use the RMAN interface to convert database files when the endian formats are different between platforms.
☒ **A, C,** and **D** are wrong since you can't use any of these interfaces or tools to convert your datafiles.

22. ☑ **A** and **D.** You may convert the datafiles either on the source or on the target system, and perform a Data Pump import afterwards.
☒ **B** and **C** are incorrect since they mention exporting of the metadata instead of importing.

23. ☑ **A.** The `V$TRANSPORTABLE_PLATFORM` view shows you all the platforms that are compatible with each other. You join this view with the `V$DATABASE` view to determine platform compatibility.
☒ **B, C,** and **D** are incorrect since there are no such views.

24. ☑ **D.** You can transport tablespaces across platforms even if the endian formats are different, as long as you convert the datafiles during the transport process.
☒ **A** is wrong since you can't automatically transport tablespaces between two platforms if they both are in the `V$TRANSPORTABLE_TABLESPACES` view. If the endian formats of the two platforms vary, you need to perform a file conversion first. **B** is incorrect because file conversions aren't mandatory for transporting tablespaces—you need to convert datafiles only if the file semantics (endian formats) are different between the two platforms, since the endian formats cannot be different for implementing transportable tablespaces. **C** is incorrect because the opposite is true—you can automatically transport tablespaces across platforms if the endian formats are *identical*.

LAB ANSWER

- You can start an interactive session and attach to a running export job by using the following command at the `expdp` prompt:

```
$ expdp salapati/sammyy1 attach=SALAPATI.SYS_EXPORT_SCHEMA_01
```

To stop the running job, you issue the following command at the `expdp` prompt (if you want an immediate rather than an orderly stoppage of the export job, you can use the command `STOP_JOB=IMMEDIATE`):

```
expdp>   STOP_JOB
```

- The name of the master table is always the same as the job name. In this case, it is SYS_EXPORT_SCHEMA.
- Once you attach to the running export session, you can issue various comamnds at the operating system prompt. To make your export session perform an export to four dump files simultaneously, you issue the following command:

```
expdp> PARALLEL=4
```

- To resume your export job after making the changes, you issue the following command:

```
expdp>  START_JOB
```

The `START_JOB` command doesn't "start" a new job—it resumes a job you stopped by attaching to it first. If you have both the dump file and the master table (which is in the export dump file), you can always resume a stopped export job without any data loss or corruption of data.