Data Pump

Data Pump replaces EXP and IMP. It provides high speed, parallel, bulk data and metadata movement of Oracle database contents across platforms and database versions. Oracle states that Data Pump's performance on data retrieval is 60% faster than Export and 20% to 30% faster on data input than Import. If a data pump job is started and fails for any reason before it has finished, it can be restarted at a later time. The commands to start the data pump are **expdb** and **impdb**, respectively. The data pump uses files as well as direct network transfer. Clients can detach and reconnect from/to the data pump. It can be monitored through several views like *dba_datapump_jobs*. The Data Pump's public API is the DBMS_DATAPUMP package.

To use Data Pump you must have EXP_FULL_DATABASE or IMP_FULL_DATABASE depending the operation to perform. These allow you to expdp & impdp across ownership for items such as grants, resource plans, schema definitions, and re-map, re-name, or re-distribute database objects or structures. By definition, Oracle gives permission to the objects in a DIRECTORY that a user would not normally have access to.
Data Pump runs only on the server side. You may initiate the export from a client but the job(s) and the files will run inside an Oracle server. There will be no dump files (expdat.dmp) or log files created on your local machine. Oracle creates dump and log files through DIRECTORY objects. So before you can use Data Pump you must create a DIRECTORY object. Example:

```
CREATE or REPLACE DIRECTORY datapump AS 'C:\user\datafile\datapump';
```

Then, as you use Data Pump you can reference this DIRECTORY as a parameter for export where you would like the dump or log files to end up.

The default name / location of Data Pump is DATA_PUMP_DIR at 'C:\oracle\product\10.2.0\admin\$ORACLE_SID\dpdump\' or /u01/app/oracle/admin/$ORACLE_SID/dpdump/

Advantages of Data Pump
1. We can perform export in parallel. It can also write to multiple files on different disks. (Specify parameters PARALLEL=2 and the two directory names with file specification DUMPFILE=ddir1:/file1.dmp, DDIR2:/file2.dmp)
2. Has the ability to attach and detach from job, monitor the job progress remotely.
3. Has more options to filter metadata objects. Ex, EXCLUDE, INCLUDE
4. ESTIMATE_ONLY option can be used to estimate disk space requirements before it performs the job
5. Data can be exported from remote database by using Database link
6. Explicit DB version can be specified, so only supported object types are exported.
7. During impdp, we can change the target file names, schema, and tablespace. Ex, REMAP_SCHEMA, REMAP_DATAFILES, REMAP_TABLESPACE
8. Has the option to filter data rows during impdp. Traditional exp/imp, we have this filter option only in exp. But here we have filter option on both impdp, expdp.
9. Data can be imported from one DB to another without writing to dump file, using NETWORK_LINK parameter.
10. Data access methods are decided automatically. In traditional exp/imp, we specify the value for the parameter DIRECT. But here, it decides where direct path can not be used , conventional path is used.
11. Job status can be queried directly from data dictionary(For example, dba_datapump_jobs, dba_datapump_sessions etc)

Equivalent exp & expdp parameters:

The below parameters are equivalent parameters between exp & expdp:

exp Command	**expdp Command**
FEEDBACK	STATUS
FILE	DUMPFILE
LOG	LOGFILE
OWNER	SCHEMAS
TTS_FULL_CHECK	TRANSPROT_FULL_CHECK

New parameters in expdp Utility

ATTACH Attach the client session to existing data pump jobs
CONTENT Specify what to export(ALL, DATA_ONLY, METADATA_ONLY)

DIRECTORY Location to write the dump file and log file.
ESTIMATE Show how much disk space each table in the export job consumes.
ESTIMATE_ONLY It estimate the space, but does not perform export
EXCLUDE List of objects to be excluded
INCLUDE List of jobs to be included
JOB_NAME Name of the export job
KEEP_MASTER Specify Y not to drop the master table after export
NETWORK_LINK Specify dblink to export from remote database
NOLOGFILE Specify Y if you do not want to create log file
PARALLEL Specify the maximum number of threads for the export job
VERSION DB objects that are incompatible with the specified version will not be exported.
ENCRYPTION_PASSWORD The table column is encrypted, then it will be written as clear text in the dump file set when the password is not specified. We can define any string as a password for this parameter.
COMPRESSION Specifies whether to compress metadata before writing to the dump file set. Values are: ALL, (METADATA_ONLY), DATA_ONLY and NONE.
SAMPLE - Allows you to specify a percentage of data to be sampled and unloaded from the source database. The sample_percent indicates the probability that a block of rows will be selected as part of the sample.
REMAP_DATA = Used to Mask (hide) data

Equivalent imp & impdp parameters

These below parameters are equivalent parameters between imp & impdp

imp Command	impdp Command
DATAFILES	TRANSPORT_DATAFILES
DESTROY	REUSE_DATAFILES
FEEDBACK	STATUS
FILE	DUMPFILE
FROMUSER	SCHEMAS, REMAP_SCHEMAS
IGNORE	TABLE_EXISTS_ACTION(SKIP,APPEND,TRUNCATE,REPLACE)
INDEXFILE, SHOW	SQLFILE
LOG	LOGFILE
TOUSER	REMAP_SCHEMA

New parameters in impdp Utility

FLASHBACK_SCN Performs import operation that is consistent with the SCN specified from the source database. Valid only when NETWORK_LINK parameter is used.
FLASHBACK_TIME Similar to FLASHBACK_SCN, but oracle finds the SCN close to the time specified.
NETWORK_LINK Performs import directly from a source database using database link name specified in the parameter. The dump file will be not be created in server when we use this parameter. To get a consistent export from the source database, we can use the FLASHBACK_SCN or FLASHBACK_TIME parameters. These two parameters are only valid when we use NETWORK_LINK parameter.
REMAP_DATAFILE Changes name of the source DB data file to a different name in the target.
REMAP_SCHEMA Loads objects to a different target schema name.
REMAP_TABLESPACE Changes name of the source tablespace to a different name in the target.
TRANSFORM We can specify that the storage clause should not be generated in the DDL for import. This is useful if the storage characteristics of the source and target database are different. The valid values are SEGMENT_ATTRIBUTES, STORAGE. STORAGE removes the storage clause from the CREATE statement DDL, whereas SEGMENT_ATTRIBUTES removes physical attributes, tablespace, logging, and storage attributes.
TRANSFORM = name:boolean_value[:object_type], where boolean_value is Y or N.
For instance, TRANSFORM=storage:N:table
ENCRYPTION_PASSWORD It is required on an import operation if an encryption password was specified on the export operation.
CONTENT, INCLUDE, EXCLUDE are same as expdp utilities.
REMAP_DATA = Used to Mask (hide) data

Data Masking with Oracle Data Pump

In addition to Data Masking Pack in Grid Control, Oracle Data Pump provides a method to mask data: REMAP_DATA parameter introduced in Oracle Database 11g.
Oracle Data Pump's REMAP_DATA feature uses a remapping function to rewrite data.
For example, a column with phone numbers could be replaced by a numbers generated by a REMAP_DATA function.

REMAP_DATA allows transformation of column's data while exporting (expdp) or importing (impdp) by using a remapping function in the database.
REMAP_DATA with Data Pump is usually faster than a custom UPDATE statement for masking data.
To mask multiple columns in the same process and command, the REMAP_DATA parameter can be used multiple times.

Restrictions
- Data types must be same in the table column, masking function parameter, and function return value.
- No commits or rollbacks should be done in the masking function.
- No direct path loads can be used in import process with REMAP_DATA.

Note: Operation of long export/import data pump processes can be monitored from the v$session_longops view, but the estimated values do not take into account REMAP_DATA operations.

Quick Example:
REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function

[schema1.]tablename.column_name:
[schema2.]pkg.function

- schema1 -- the schema with the table to be remapped. By default, this is the schema of the user doing the export.
- tablename -- the table which column will be remapped.
- column_name -- which is to be remapped.
- schema2 -- with the PL/SQL package for remapping function. As a default, this is the schema of the user doing the export.
- pkg -- the name of the PL/SQL package with the remapping function.
- function -- the name of the remap function in the PL/SQL package

Create a table in the CUSTOMERS schema called phones

```
CREATE TABLE CUSTOMERS.PHONES (
MODELNAME VARCHAR(20) NOT NULL,
PHONENUMBER VARCHAR2(50));
insert into CUSTOMERS.PHONES values('N900','+3581234567');
insert into CUSTOMERS.PHONES values('N8','+3589817654');
insert into CUSTOMERS.PHONES values('N7','+3584834819');
Commit;
```

We then need to create a function for remapping. The function masknumber will accept a varchar2 type and returns a random phone number in varchar2 type

```
create or replace package customers.maskpkg
as
     function masknumber(phonenumber varchar2) return varchar2;
end;
/
create or replace package body customers.maskpkg as
   function masknumber (phonenumber varchar2) return varchar2 is
   begin
      return substr(phonenumber,1,4)||round(dbms_random.value (100,999))|| lpad(round(dbms_random.value
(1,9999)),4,'0');
   end;
end;
/
```

This example will mask one column: phonenumber to the export output file

```
expdp customers tables=customers.phones dumpfile=phones_masked.dmp directory=dumpdir
remap_data=customers.phones.phonenumber:customers.maskpkg.masknumber
```

REMAP_DATA parameter can also be used in the import process

```
impdp dumpfile=data.dmp REMAP_DATA=scott.orders.customer_name:scott.maskpkg.mask
```

```
expdp dumpfile=data.dmp REMAP_DATA=scott.orders.customer_name:scott.maskpkg.mask
```

Compression

Usually, when you have to use expdp you need to compress/uncompress the exported file, adding more time and also taking more CPU time. In oracle 11g, there is no need to use OS-level compress utility, the COMPRESSION option not only is faster, it also makes the file much smaller. Valid values are: ALL, (METADATA_ONLY), DATA_ONLY and NONE.

Example:
nohup expdp system schemas=test directory=data_pump_dir dumpfile=test.dmp logfile=test.log compression=all &

Some Examples:

In all this cases I will be using ORCL as the original DB, and DEST as the destination DB.

Scenario1 Export the whole ORCL database.
expdp userid=system/password@ORCL dumpfile=expfulldp.dmp logfile=expfulldp.log **full**=y directory=dumplocation

Scenario2 Export the scott schema from ORCL and import into DEST database.
expdp userid=system/password@ORCL dumpfile=schemaexpdb.dmp logfile=schemaexpdb.log directory=dumplocation **schemas**=scott
impdp userid=system/password@DEST dumpfile=schemaexpdb.dmp logfile=schemaimpdb.log directory=dumplocation

Another Example: While import, exclude some objects(sequence,view,package,cluster,table). Load the objects which came from RES tablespace into USERS tablespace in target database.
impdp userid=system/password@DEST dumpfile=schemaexpdb.dmp logfile=schemaimpdb.log directory=dumplocation
table_exists_action=replace **remap_tablespace**=res:users **exclude**=sequence,view,package,cluster,table:"in('LOAD_EXT')"

Scenario 3 Clone a User
In the past when a DBA had the need to create a new user with the same structure (All objects, tablespaces quota, synonyms, grants, system privileges, etc) was a very painful experience, now all can be done very easily using Data Pump, let use as an example that you want to create the user "Z" exactly like the user "A", to achieve this goal all you will need to do is first export the schema "A" definition and then import it again saying to the Data Pump to change the schema "A" for the new schema named "Z" using the "remap_schema" parameter available with impdp.
expdp user/password **schemas**=A directory=datapump dumpfile=Schema_A.dmp [optional: **content**=metadata_only]
impdp user/password **remap_schema**=A:Z directory=datapump dumpfile= Schema_A.dmp
And your new user Z is now created like your existing user A , that easy!

Scenario 4 Create a Metadata File
You can generate a SQL File from an existing exported file. As an Example, I am going to expdp the Schema Fraudguard, after that, I will use the impdp command with the sqlfile option to generate a sql containing all the objects that I already exported:
expdp system schemas=fraudguard **content**=metadata_only directory=EXPORTPATH dumpfile=metadata_24112010.dmp
impdp system directory=EXPORTPATH dumpfile= metadata_24112010.dmp **sqlfile**=metadata_24112010.sql

Scenario5 Export the emp table from scott schema at ORCL instance and import into DEST instance.
expdp userid=system/password@ORCL logfile=tableexpdb.log directory=dumplocation **tables**=scott.part_emp dumpfile=tableexpdb.dmp
impdp userid=system/password@DEST dumpfile=tableexpdb.dmp logfile=tabimpdb.log directory=dumplocation
table_exists_action=REPLACE

Scenario 6 Create smaller Copies of PROD
That is a very common task for a DBA, you have a task to create a copy of your Database (for development or test purpose) but your destination server don't have enough space to create a full copy of it!
This can be easily solved with Data Pump, for this example, let say that you only have space for 70% of your production database, now to know how to proceed, we need to decide if the copy will contain metadata only (no data/rows) or if it will include the data also. Let's see how to do each way:

a) Metadata Only
First do a full export of your source database.
```
expdp user/password content=metadata_only full=y directory=datapump dumpfile=metadata_24112010.dmp
```

Then, let's import the metadata and tell the Data Pump to reduce the size of extents to 70%, you can do it using the parameter "transform" available with "impdp", it represent the percentage multiplier that will be used to alter extent allocations and datafiles size.

```
impdp user/password transform=pctspace:70 directory=datapump dumpfile=metadata_24112010.dmp
```

b) Metadata and data

First does a full export of your source database using the export parameter "sample", this parameter specify a percentage of the data rows to be sampled and unload from your source database, in this case let's use 70%.

```
expdp user/password sample=70 full=y directory=datapump dumpfile=expdp_70_24112010.dmp
```

Then, all you need to do as the example before is to import it telling the Data Pump to reduce the size of extents to 70%, and that's it!

```
impdp user/password transform=pctspace:70 directory=datapump dumpfile=expdp_70_24112010.dmp
```

Scenario 7 Export only specific partition in emp table from scott schema at orcl and import into ordb database.

expdp userid=system/password@ORCL dumpfile=partexpdb.dmp logfile=partexpdb.log directory=dumplocation **tables**=scott.part_emp:part10,scott.part_emp:part20

If we want to overwrite the exported data in target database, then we need to delete emp table for deptno in(10,20).

scott@DEST> delete part_emp where deptno in (10,20);
scott@DEST> commit;
impdp userid=system/password@DEST dumpfile=partexpdb.dmp logfile=tabimpdb.log directory=dumplocation **table_exists_action**=append

Scenario 8 Export only tables (no code) in scott schema at ORCL and import into DEST database

expdp userid=system/password@ORCL dumpfile=schemaexpdb.dmp logfile=schemaexpdb.log directory=dumplocation **include**=table **schemas**=scott
impdp userid=system/password@DEST dumpfile=schemaexpdb.dmp logfile=schemaimpdb.log directory=dumplocation **table_exists_action**=replace

Scenario 9 Export only rows belonging to department 10 and 20 in emp and dept table from ORCLdatabase. Import the dump file in @DESTdatabase. While importing, load only deptno 10 in target database.

expdp userid=system/password@ORCL dumpfile=data_filter_expdb.dmp logfile=data_filter_expdb.log directory=dumplocation **content**=data_only **schemas**=scott **include**=table:"in('EMP','DEPT')" **query**="where deptno in(10,20)"

impdp userid=system/password@DEST dumpfile=data_filter_expdb.dmp logfile=data_filter_impdb.log directory=dumplocation **schemas**=scott **query**="where deptno = 10" **table_exists_action**=APPEND

Scenario 10 Export the scott schema from ORCLdatabase and split the dump file into 50M sizes. Import the dump file into DEST datbase.

Expdp parfile content:
userid=system/password@ORCL
logfile=schemaexp_split.log
directory=dumplocation
dumpfile=schemaexp_split_%U.dmp
filesize=50M
schemas=scott
include=table

As per the above expdp parfile, initially, schemaexp_split_01.dmp file will be created. Once the file is 50MB, the next file called schemaexp_split_02.dmp will be created. Let us say, the dump file size is 500MB, then it creates 10 dump file as each file size is 50MB.
Impdp parfile content:
userid=system/password@DEST
logfile=schemaimp_split.log
directory=dumplocation
dumpfile=schemaexp_split_%U.dmp
table_exists_action=replace
remap_tablespace=res:users
exclude=grant

Scenario 11 Export the scott schema from ORCL database and split the dump file into four files. Import the dump file into DEST datbase.

Expdp parfile content:
userid=system/password@ORCL
logfile=schemaexp_split.log
directory=dumplocation
dumpfile=schemaexp_split_%U.dmp

```
parallel=4
schemas=scott
include=table
```

As per the above parfile content, initially four files will be created - schemaexp_split_01.dmp, schemaexp_split_02.dmp, schemaexp_split_03.dmp, schemaexp_split_04.dmp. Notice that every occurrence of the substation variable is incremented each time. Since there is no FILESIZE parameter, no more files will be created.
Impdp parfile content:

```
userid=system/password@DEST
logfile=schemaimp_split.log
directory=dumplocation
dumpfile=schemaexp_split_%U.dmp
table_exists_action=replace
remap_tablespace=res:users
exclude=grant
```

Scenario 12 Export the scott schema from ORCL database and split the dump file into three files.

The dump files will be stored in three different location. This method is especially useful if you do not have enough space in one file system to perform the complete expdp job. After export is successful, import the dump file into DEST database.
Expdp parfile content:

```
userid=system/password@ORCL
logfile=schemaexp_split.log
directory=dumplocation
dumpfile=dump1:schemaexp_%U.dmp,dump2:schemaexp_%U.dmp,dump3:schemaexp_%U.dmp
filesize=50M
schemas=scott
include=table
```

As per above expdp par file content, it place the dump file into three different location. Let us say, entire expdp dump file size is 1500MB. Then it creates 30 dump files(each dump file size is 50MB) and place 10 files in each file system.
Impdp parfile content:

```
userid=system/password@DEST
logfile=schemaimp_split.log
directory=dumplocation
dumpfile=dump1:schemaexp_%U.dmp,dump2:schemaexp_%U.dmp,dump3:schemaexp_%U.dmp
table_exists_action=replace
```

Scenario 13 Expdp scott schema in ORCL and impdp the dump file in training schema in DEST database.

```
expdp userid=scott/tiger@ORCL logfile=netwrokexp1.log directory=dumplocation dumpfile=networkexp1.dmp schemas=scott
include=table

impdp userid=system/password@DEST logfile=networkimp1.log directory=dumplocation dumpfile=networkexp1.dmp
table_exists_action=replace remap_schema=scott:training
```

Scenario 14 Expdp table on ORCL database and imdp in DEST. When we export the data, export only 20 percent of the table data. We use SAMPLE parameter to accomplish this task.

SAMPLE parameter allows you to export subsets of data by specifying the percentage of data to be sampled and exported. The sample_percent indicates the probability that a block of rows will be selected as part of the sample. It does not mean that the database will retrieve exactly that amount of rows from the table. The value you supply for sample_percent can be anywhere from .000001 up to, but not including, 100.
If no table is specified, then the sample_percent value applies to the entire export job. The SAMPLE parameter is not valid for network exports.

```
expdp userid=system/password@ORCL dumpfile=schemaexpdb.dmp logfile=schemaexpdb.log directory=dumplocation
tables=scott.part_emp SAMPLE=20
```

As per the above expdp parfile, it exports only 20 percent of the data in part_emp table.

```
impdp userid=system/password@DEST dumpfile=schemaexpdb.dmp logfile=schemaimpdb.log directory=dumplocation
table_exists_action=replace
```

Exclude Objects
The exclude parameter allows any database object type to be excluded from an export or import operation. The optional name qualifier allows you finer selectivity within each object type specified. For example, the following three lines in a parameter file:
Exclude=function
Exclude=procedure
Exclude=package:"like 'PAYROLL%' "
Would exclude all functions, procedures and packages with names starting with 'PAYROLL' from the job.

Include
The include parameter includes only the specified object types and objects in an operation. For example, if the above three specifications were INCLUDE parameters in a full database export, only functions, procedures and packages with names starting with 'PAYROLL' would be written to the dumpfile set. You can use the clause INCLUDE=TABLE:"LIKE 'TAB%'" to export only those tables whose name start with TAB. Similarly, you could use the construct INCLUDE=TABLE:"NOT LIKE 'TAB%'" to exclude all tables starting with TAB. Alternatively you can use the EXCLUDE parameter to exclude specific objects.

Content
The content parameter allows one to request for the current operation just metadata, just data or both. The exp 'ROWS=N' parameter is equivalent to content=metadata_only, but there is no equivalent for content=data_only.

Query
The query parameter operates much as it did in original export, but with two significant enhancements:
1. It can be qualified with a table name such that it only applies to that table
2. It can be used during import as well as export.

table_exists_action
The TABLE_EXISTS_ACTION=APPEND parameter allows data to be imported into existing tables. Other uses:
 * skip
 * append
 * truncate
 * replace

I/O BANDWIDTH IS MOST IMPORTANT FACTOR
It is important to make sure there is sufficient I/O bandwidth to handle the number of parallel threads specified; otherwise performance can actually degrade with additional parallel threads. Care should be taken to make sure the dumpfile set is located on spindles other than those holding the instance's data files. Wildcard file support makes it easy to spread the I/O load over multiple spindles. For example, a specification such as:
Dumpfile=dmpdir1:full1%u.dmp,dmpdir2:full2%u.dmp
Dumpfile=dmpdir3:full3%u.dmp,dmpdir4:full4%u.dmp
will create files named full101.dmp, full201.dmp, full301.dmp, full401.dmp, full102.dmp, full202.dmp, full302.dmp, etc. in a round-robin fashion across the four directories pointed to by the four directory objects.
In parallel mode, the status screen will show four worker processes. (In default mode, only one process will be visible.) All worker processes extract data simultaneously and show their progress on the status screen

INITIALIZATION PARAMETERS
Essentially no tuning is required to achieve maximum Data Pump performance. Initialization parameters should be sufficient out of the box.

- Make sure **disk_asynch_io** remains TRUE: It has no effect on those platforms whose file systems already support asynchronous I/O, but those that don't are significantly impacted by a value of FALSE.
- **db_block_checksum**'s default value is FALSE, but if an integrity issue is being investigated requiring this to be set to TRUE, its impact on data loading and unloading is minimal; less than 5%.

Oracle recommends the following settings to improve performance.
Disk_Asynch_io= true
Db_block_checking=false
Db_block_checksum=false

Additionally, the number of processes and sessions allowed to the database must be set to high, to allow for maximum parallelism.

More Examples
To export only a few specific objects--say, function LIST_DIRECTORY and procedure DB_MAINTENANCE_DAILY--you could use

```
expdp ananda/iclaim directory=DPDATA1 dumpfile=expprocs.dmp
include=PROCEDURE:\"=\'DB_MAINTENANCE_DAILY\'\",FUNCTION:\"=\'LIST_DIRECTORY\'\"
```

This dumpfile serves as a backup of the sources. You can even use it to create DDL scripts to be used later. A special parameter called SQLFILE allows the creation of the DDL script file.
This instruction creates a file named procs.sql in the directory specified by DPDATA1, containing the scripts of the objects inside the export dumpfile. This approach helps you create the sources quickly in another schema.

```
impdp ananda/iclaim directory=DPDATA1 dumpfile=expprocs.dmp sqlfile=procs.sql
```

Export/Import a few tables:

```
expdp scott/tiger tables=EMP,DEPT directory=TEST_DIR dumpfile=EMP_DEPT.dmp logfile=expdpEMP_DEPT.log
impdp scott/tiger tables=EMP,DEPT directory=TEST_DIR dumpfile=EMP_DEPT.dmp logfile=impdpEMP_DEPT.log
```

The OWNER parameter of exp has been replaced by the SCHEMAS parameter which is used to specify the schemas to be exported. The following is an example of the schema export and import syntax:

```
expdp scott/tiger schemas=SCOTT directory=TEST_DIR dumpfile=SCOTT.dmp logfile=expdpSCOTT.log
impdp scott/tiger schemas=SCOTT directory=TEST_DIR dumpfile=SCOTT.dmp logfile=impdpSCOTT.log
```

The REMAP_TABLESPACE in the impdp sencence allows you to move the objects from one tablespace to another one.

```
impdp system SCHEMAS=SCOTT directory=EXPORTPATH DUMPFILE=SCOTT.dmp LOGFILE=imp.log
REMAP_TABLESPACE=FGUARD_DATA:FG_DATA
```

You can also use several REMAP_TABLESPACE clauses in the impdp sencence:

```
impdp system SCHEMAS=SCOTT directory=EXPORTPATH DUMPFILE=SCOTT.dmp LOGFILE=imp.log
REMAP_TABLESPACE=FGUARD_DATA:FG_DATA remap_tablespace=FGUARD_INDX:FG_INDX
```

The FULL parameter indicates that a complete database export is required. The following is an example of the full database export and import syntax:

```
expdp system/password full=Y directory=TEST_DIR dumpfile=DB10G.dmp logfile=expdpDB10G.log
impdp system/password full=Y directory=TEST_DIR dumpfile=DB10G.dmp logfile=impdpDB10G.log
```

Data pump performance can be improved by using the PARALLEL parameter. This should be used in conjunction with the "%U" wildcard in the DUMPFILE parameter to allow multiple dumpfiles to be created or read:

```
expdp scott/tiger schemas=SCOTT directory=TEST_DIR parallel=4 dumpfile=SCOTT_%U.dmp
logfile=expdpSCOTT.log
```

Each thread creates a separate dumpfile, so the parameter dumpfile should have as many entries as the degree of parallelism.
Note how the dumpfile parameter has a wild card %U, which indicates the files will be created as needed and the format will be SCOTT_nn.dmp, where nn starts at 01 and goes up as needed.

The INCLUDE and EXCLUDE parameters can be used to limit the export/import to specific objects. When the INCLUDE parameter is used, only those objects specified by it will be included in the export. When the EXCLUDE parameter is used all objects except those specified by it will be included in the export:

```
expdp scott/tiger schemas=SCOTT include=TABLE:\"IN (\'EMP\', \'DEPT\')\" directory=TEST_DIR
dumpfile=SCOTT.dmp logfile=expdpSCOTT.log
expdp scott/tiger schemas=SCOTT exclude=TABLE:\"= \'BONUS\'\" directory=TEST_DIR dumpfile=SCOTT.dmp
logfile=expdpSCOTT.log
```

Monitoring Export:

While Data Pump Export is running, press Control-C; it will stop the display of the messages on the screen, but not the export process itself. Instead, it will display the Data Pump Export prompt as shown below. The process is now said to be in "interactive" mode:

```
Export>
```

This approach allows several commands to be entered on that Data Pump Export job. To find a summary, use the STATUS command at the prompt:

```
Export> status
Job: CASES_EXPORT
  Operation: EXPORT
  Mode: TABLE
  State: EXECUTING
  Degree: 1
  Job Error Count: 0
  Dump file:  /u02/dpdata1/expCASES.dmp
      bytes written =  2048
```

```
Worker 1 Status:
  State: EXECUTING
  Object Schema: DWOWNER
  Object Name: CASES
  Object Type: TABLE_EXPORT/TBL_TABLE_DATA/TABLE/TABLE_DATA
  Completed Objects: 1
  Total Objects: 1
  Completed Rows: 4687818
```

Remember, this is merely the status display. The export is working in the background. To continue to see the messages on the screen, use the command CONTINUE_CLIENT from the Export prompt.
While Data Pump jobs are running, you can pause them by issuing STOP_JOB on the Data Pump Export or Data Pump Import prompts and then restart them with START_JOB.
This functionality comes in handy when you run out of space and want to make corrections before continuing.

A simple way to gain insight into the status of a Data Pump job is to look into a few views maintained within the Oracle instance the Data Pump job is running.
These views are DBA_DATAPUMP_JOBS, DBA_DATAPUMP_SESSIONS, and V$SESSION_LOGOPS and they are critical in the monitoring of your export jobs so, you can attach to a Data Pump job and modify the execution of the that job.

DBA_DATAPUMP_JOBS
This view will show the active Data Pump jobs, their state, degree of parallelism, and the number of sessions attached.

```
select * from dba_datapump_jobs
OWNER_NAME JOB_NAME               OPERATION  JOB_MODE   STATE         DEGREE    ATTACHED_SESSIONS
---------- ---------------------- ---------- ---------- ------------- --------- -----------------
JKOOP      SYS_EXPORT_FULL_01     EXPORT     FULL       EXECUTING     1          1
JKOOP      SYS_EXPORT_SCHEMA_01   EXPORT     SCHEMA     EXECUTING     1          1
```

DBA_DATAPUMP_SESSIONS
This view give gives the SADDR that assist in determining why a Data Pump session may be having problems. Join to the V$SESSION view for further information.

```
SELECT * FROM DBA_DATAPUMP_SESSIONS
OWNER_NAME JOB_NAME                       SADDR
---------- ------------------------------ --------
JKOOPMANN  SYS_EXPORT_FULL_01             225BDEDC
JKOOPMANN  SYS_EXPORT_SCHEMA_01           225B2B7C
```

V$SESSION_LONGOPS
This view helps determine how well a Data Pump export is doing. It also shows you any operation that is taking long time to execute. Basically gives you a progress indicator through the MESSAGE column.

```
select username, opname, target_desc,
       sofar, totalwork, message
from V$SESSION_LONGOPS

USERNAME OPNAME               TARGET_DES SOFAR TOTALWORK  MESSAGE
-------- -------------------- ---------- ----- ---------- ------------------------------------------------
JKOOP    SYS_EXPORT_FULL_01   EXPORT       132        132 SYS_EXPORT_FULL_01:EXPORT:132 out of 132 MB done
JKOOP    SYS_EXPORT_FULL_01   EXPORT        90        132 SYS_EXPORT_FULL_01:EXPORT:90 out of 132 MB done
JKOOP    SYS_EXPORT_SCHEMA_01 EXPORT        17         17 SYS_EXPORT_SCHEMA_01:EXPORT:17 out of 17 MB done
JKOOP    SYS_EXPORT_SCHEMA_01 EXPORT        19         19 SYS_EXPORT_SCHEMA_01:EXPORT:19 out of 19 MB done
```

Another example:
We want to check how much a specific session (sid=9) needs to perform in order to finish. So using the PRINT_TABLE function described in AskTom.com you we can do the following:

```
set serveroutput on size 999999
exec print_table('select * from v$session_longops where sid = 9');

SID                           : 9
SERIAL#                       : 68
OPNAME                        : Transaction Rollback
TARGET                        :
TARGET_DESC                   : xid:0x000e.01c.00000067
SOFAR                         : 10234
TOTALWORK                     : 20554
```

```
UNITS                          : Blocks
START_TIME                     : 07-dec-2003 21:20:07
LAST_UPDATE_TIME               : 07-dec-2003 21:21:24
TIME_REMAINING                 : 77
ELAPSED_SECONDS                : 77
CONTEXT                        : 0
MESSAGE                        : Transaction Rollback: xid:0x000e.01c.00000067 :
                                  10234 out of 20554 Blocks done
USERNAME                       : SYS
SQL_ADDRESS                    : 00000003B719ED08
SQL_HASH_VALUE                 : 1430203031
SQL_ID                         : 306w9c5amyanr
QCSID                          : 0
```

Let's examine each of these columns carefully. There may be more than one long running operation in the session—especially because the view contains the history of all long running operations in previous sessions. The column OPNAME shows that this record is for "Transaction Rollback," which points us in the right direction. The column TIME_REMAINING shows the estimated remaining time in seconds, described earlier and the column ELAPSED_SECONDS shows the time consumed so far. So how does this table offer an estimate of the remaining time? Clues can be found in the columns TOTALWORK, which shows the total amount of "work" to do, and SOFAR, which shows how much has been done so far. The unit of work is shown in column UNITS. In this case, it's in blocks; therefore, a total of 10,234 blocks have been rolled back so far, out of 20,554. The operation so far has taken 77 seconds. Hence the remaining blocks will take: 77 * (10234 / (20554-10234)) ≈ 77 seconds But you don't have to take that route to get the number; it's shown clearly for you. Finally, the column LAST_UPDATE_TIME shows the time as of which the view contents are current, which will serve to reinforce your interpretation of the results.

Which object types are left?

```
SQL> select unique object_type_seqno, object_type
       from system.sys_import_full_01
       where process_order > 0 AND processing_state = ‘R’
         and processing_status = ‘C’;

OBJECT_PATH_SEQNO OBJECT_TYPE
----------------- -----------------------------
              103 PROCEDURE
              119 ALTER_PROCEDURE
              137 VIEW
```

What is left for the current object?

```
SQL> select object_schema, object_name
       from system.sys_import_full_01
       where process_order > 0 and processing_state = 'R'
         and processing_status = 'C'
         and object_path_seqno = 103;

OBJECT_SCHEMA OBJECT_NAME
-------------------- --------------------
                  HR ADD_JOB_HISTORY
                  HR SECURE_DML
```

Another Example using PL/SQL and the Data Pump API:

The Data Pump API, DBMS_DATAPUMP, provides a high-speed mechanism to move the data from one database to another. The structure used in the client interface of this API is a job handle. Job handle can be created using the OPEN or ATTACH function of the DBMS_DATAPUMP package. Other DBA sessions can attach to a job to monitor and control its progress so that remote DBA can monitor the job that was scheduled by an on-site DBA.
The following steps list the basic activities involved in using Data Pump API.

1. Execute DBMS_DATAPUMP.OPEN procedure to create job.
2. Define parameters for the job like adding file and filters etc.
3. Start the job.
4. Optionally monitor the job until it completes.
5. Optionally detach from job and attach at later time.
6. Optionally, stop the job
7. Restart the job that was stopped.

```
declare
h1   NUMBER;
 begin
  begin
      h1 := dbms_datapump.open (operation => 'IMPORT', job_mode => 'SCHEMA', job_name => 'IMPORTGO',
version => 'COMPATIBLE');
  end;
  begin
     dbms_datapump.set_parallel(handle => h1, degree => 1);
  end;
  begin
     dbms_datapump.add_file(handle => h1, filename => 'IMPORT.LOG', directory => 'DATAPUMP', filetype =>
3);
  end;
  begin
     dbms_datapump.set_parameter(handle => h1, name => 'KEEP_MASTER', value => 0);
  end;
  begin
     dbms_datapump.add_file(handle => h1, filename => 'FRAUDGUARD.dmp', directory => 'DATAPUMP', filetype
=> 1);
  end;
  begin
     dbms_datapump.metadata_remap(handle => h1, name => 'REMAP_SCHEMA', old_value => 'FRAUDGUARD', value
=> 'PROD_FNT_FG83540');
  end;
  begin
     dbms_datapump.metadata_remap(handle => h1, name => 'REMAP_TABLESPACE', old_value => 'FGUARD_DATA',
value => 'FG_DATA');
  end;
  begin
     dbms_datapump.metadata_remap(handle => h1, name => 'REMAP_TABLESPACE', old_value =>
'FGUARD_ARCH_DATA', value => 'FG_DATA');
  end;
  begin
     dbms_datapump.metadata_remap(handle => h1, name => 'REMAP_TABLESPACE', old_value => 'FGUARD_INDX',
value => 'FG_INDX');
  end;
  begin
     dbms_datapump.metadata_remap(handle => h1, name => 'REMAP_TABLESPACE', old_value =>
'FGUARD_ARCH_INDX', value => 'FG_INDX');
  end;
  begin
     dbms_datapump.metadata_filter(handle => h1, name => 'SCHEMA_EXPR', value => 'IN(''FRAUDGUARD'')');
  end;
  begin
     dbms_datapump.set_parameter(handle => h1, name => 'DATA_ACCESS_METHOD', value => 'AUTOMATIC');
  end;
  begin
     dbms_datapump.set_parameter(handle => h1, name => 'INCLUDE_METADATA', value => 1);
  end;
  begin
     dbms_datapump.set_parameter(handle => h1, name => 'TABLE_EXISTS_ACTION', value => 'TRUNCATE');
  end;
  begin
     dbms_datapump.set_parameter(handle => h1, name => 'SKIP_UNUSABLE_INDEXES', value => 0);
  end;
  begin
     dbms_datapump.start_job(handle => h1, skip_current => 0, abort_step => 0);
  end;
  begin
     dbms_datapump.detach(handle => h1);
  end;
end;
/
```

More Examples HERE