EXAM OBJECTIVE: TRANSPORT TABLESPACES ACROSS DIFFERENT PLATFORMS

TRANSPORTABLE TABLESPACES OVERVIEW

The transportable tablespaces feature introduced in Oracle 8i, is a mechanism to move data or tablespaces across platforms. In earlier versions of Oracle this feature had many restrictions, some of which have been overcome in 10g.

In earlier versions a tablespace could be moved from one database to another provided the databases were on the same platform. In Oracle 10g, you can transport data files from one platform to another, provided the source and target databases are running on one of the platforms listed below:

```
SQL> SELECT * FROM v$transportable_platform
        ORDER BY platform_id;
```

PLATFORM_ID	PLATFORM_NAME	ENDIAN_FORMAT
1	Solaris[tm] OE (32-bit)	Big
2	Solaris[tm] OE (64-bit)	Big
3	HP-UX (64-bit)	Big
4	HP-UX IA (64-bit)	Big
5	HP Tru64 UNIX	Little
6	AIX-Based Systems (64-bit)	Big
7	Microsoft Windows IA (32-bit)	Little
8	Microsoft Windows IA (64-bit)	Little
9	IBM zSeries Based Linux	Big
10	Linux IA (32-bit)	Little
11	Linux IA (64-bit)	Little
12	Microsoft Windows 64-bit for AMD	Little
13	Linux 64-bit for AMD	Little
15	HP Open VMS	Little
16	Apple Mac OS	Big

- Both the source and target databases need to set the COMPATIBLE initialization parameter to 10.0.0 or higher. This will cause all files opened to become platform aware.
- All tablespaces that are read-only prior to Oracle 10g must be made read-write before using this feature.

Steps for configuring Transportable Tablespaces

The basic steps are similar to what was done in earlier releases except when both platforms use different endian formats.

1. Make the tablespaces read-only
2. Use Data Pump to extract the metadata
3. Check if target has the same endian format – If yes, go to step 5 else go to step 4.
4. Convert the datafile using RMAN – (this step is done so that the tablespace can be understood by the target database). The CONVERT command of the RMAN utility performs byte ordering when transporting to a different endian. This step can be done on the source or the target database.
5. Ship the datafiles and dump file to the target
6. On the target database, use Data Pump to Import metadata
7. On the target database, make the tablespace read-write.

Data Dictionary changes related to Transportable tablespaces

V$DATABASE : Now includes the PLATFORM_ID and PLATFORM_NAME columns.

V$TRANSPORTABLE_PLATFORM : Is a view that lists all platforms that currently support the feature. Relevant columns include PLATFORM_ID, PLATFORM_NAME and ENDIAN_FORMAT. You can determine endianness by executing a query of the following kind:

```
SQL> SELECT endian_format
    FROM v$transportable_platform tp, v$database d
    WHERE tp.platform_name = d.platform_name;
    ENDIAN_FORMAT
    --------------
    Little
```

The RMAN Convert Command

Consider a source machine (SRC) running Linux (little endian) and the target database (TRG) running HP-UX(big endian). RMAN would convert the datafile from Linux to HP-UX format on the source machine.

Examples

```
RMAN> CONVERT TABLESPACE users
        TO PLATFORM 'HP-UX (64-bit)'
        FORMAT = '/home/oracle/rman_backup/%N_%f' ;
```

N=> tablespace name
f => datafile number
The destination file will have a name of USERS_4, if the datafile number is 0004.

```
RMAN> CONVERT TABLESPACE users, tools
  TO PLATFORM 'HP-UX (64-bit)'
    FORMAT = '/home/oracle/rman_backup.%N_%f'
    PARALLELISM = 5;
```

If users tablespace had two datafiles : users01.dbf and users02.dbf

If tools tablespace had two datafiles : tools01.dbf and tools02.dbf

The convert in the second example, would be done in parallel by 4 channels creating files USERS_4, USERS_5, TOOLS_6, TOOLS_7.

If the source filename should be retained then:

```
RMAN> CONVERT TABLESPACE users
    TO PLATFORM 'HP-UX(64-bit)'
    DB_FILE_NAME_CONVERT '/source/users/','/home/oracle/rman_backups'
        ;
```

The destination file will be created as /home/oracle/rman_backups/users01.dbf and /home/oracle/rman_backups/users02.dbf if there are two files belonging to the USERS tablespace.

You could perform this conversion on the target platform. You may do this to have a shorter downtime on the source database. You could copy the users01.dbf to the target database and issue the command

```
RMAN> CONVERT DATAFILE '/target/users/users01.dbf'
  FORMAT '/home/oracle/rman_backups/%N_%f' ;
```

EXAM OBJECTIVE: DATA PUMP ARCHITECTURE

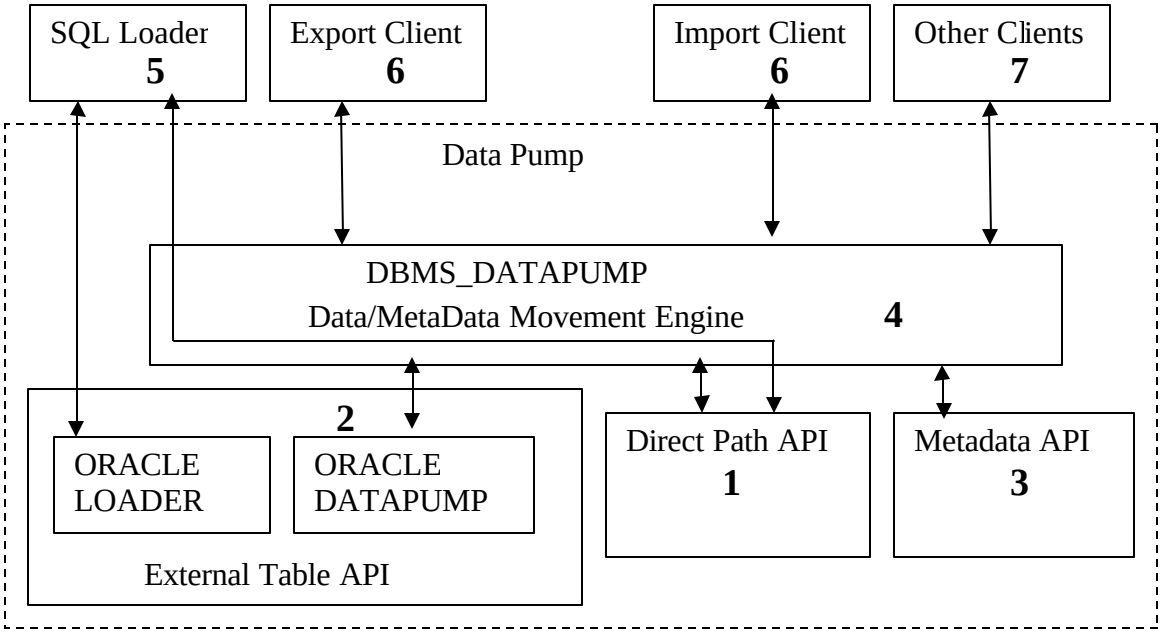
Oracle Data Pump is a new feature of Oracle 10g, that provided high speed, parallel, bulk data and metadata movement of Oracle database contents. Oracle Data Pump uses proprietary APIs to load and unload data instead of usual SQL commands. This is significantly faster.

A new interface PL/SQL package called **DBMS_DATAPUMP** provides the server-side infrastructure for this feature.

In Oracle 10g, the Export is done using the ***expdp*** and Import by ***impdp*** interface. Data Pump is server-based, rather than client-based.

All files generated such as dump files, log files and SQL files are accessed using server-side directory paths. This is so that appropriate file security can be enforced. The Enterprise Manager can also be used to perform operations provided by Data Pump.

General Architecture



Given below are the major Data Pump components and consumers:

1. **DIRECT PATH API (DPAPI)**
2. **EXTERNAL TABLE SERVICES**: Data pump uses external tables by using the drivers.
    - **ORACLE_LOADER** access driver: Provides read-only access to SQL loader compatible files and external tables.
    - **ORACLE_DATAPUMP** access driver: Provides external tables with read and write access to files containing proprietary format: binary DPAPI streams.
3. The **DBMS_METADATA** package provides database object definitions to an export process for the entire database or a requested subset in proper creation order. This package can also re-create all objects from their XML representations at import time.
4. The **DBMS_DATAPUMP** package embodies the API for high-speed export and import utilities for bulk data and metadata movement.

5. SQL*Loader client has been integrated with external tables, thereby providing automatic migration of loader control files to external table access parameters.
6. New export and import clients (**expdp** and **impdp**), make calls to the DBMS_DATAPUMP package to initiate and monitor Data Pump operations.
7. Applications such as Enterprise Manager, replication, transportable tablespaces, user applications and so on, benefit from this infrastructure.

EXAM OBJECTIVE: MONITORING DATA PUMP OPERATIONS

You can join dictionary views shown to retrieve Data Pump information. The relevant data dictionary views are:

- DBA_DATAPUMP_JOBS
- DBA_DATAPUMP_SESSIONS
- V$SESSION
- V$SESSION_LONGOPS.

EXAM OBJECTIVE: Use Data Pump Export and Import

PERFORMING DATA PUMP EXPORT

1. The first step in using Data Pump is to create a directory object. This path will determine what directories on your system will be used to locate files that are created by Data Pump.
   SQL> CREATE DIRECTORY dir1 AS '/home/oracle/dp';
2. The user who is using Data Pump export should be granted read and write permissions on these directories.
   SQL> GRANT READ, WRITE ON DIRECTORY dir1 TO userA;
3. a) TABLE MODE EXPORT: Export the data, using the OS utility expdp. This example is a table mode export using Data Pump, indicated by the TABLES parameter.

   **$ expdp userA/userA tables=EMPLOYEES **
   **directory=DIR1 **
   **dumpfile = EMPLOYEES.dmp **
   job_name=EMPLOYEE_EXPORT

   b) SCHEMA MODE Export:
   In this mode of export, objects belonging to schemas are unloaded. This is the default mode. You need to specify the SCHEMAS parameter. If a user has the EXP_FULL_DATABASE role, a list of schemas can be specified. In the example below two dumpfiles will get created by the names of sch101.dmp and sch201.dmp.

   **$ expdp system/manager SCHEMA=userA **
   DUMPFILE = dir1:sch1%U.dmp, dir2:sch2%U.dmp
   **LOGFILE= expschema.log **

```
    PARALLEL=2
    JOB_NAME= Schema_export
```

c) FULL EXPORT

In this mode of export, all data and metadata in the database are exported. Full database export is done using the keyword FULL. It can be done by a user who has DBA privileges or has the EXP_FULL_DATABASE role.

```
$ expdp system/manager FULL=y   \
  DIRECTORY=dir1  \
  DUMPFILE = fullexp.dmp   \
  JOB_NAME = fullexport
```

Important Parameters that may be specified

- **NOLOGFILE** – Values (Y/N) indicates that the export log file will not be generated.
- **ESTIMATE_ONLY** – estimates the space that would be consumed in a schema export. This parameter does not actually perform the export. The estimate is printed in a log file and printed on the client's standard input device. The estimate takes into consideration, table row data only and not metadata.
- **INCLUDE** – This parameter gives you the ability to select specific objects to be selected during the export. For instance you can choose to export only row data without any metadata, or only procedures and nothing else.
- **STOP_JOB** – to pause jobs when Data Pump jobs are running. You would need to get into interactive mode by pressing CTRL+C from a terminal window to be able to issue a command to stop the job.
- **START_JOB** – to restart a job that was paused. You may want to stop and start a job if you run out of space or want to make some kind of correction.

  Examples

  ```
  INCLUDE=PROCEDURE or
  INCLUDE= table: \"IN \( \'TAB1\',\'TAB2\'\) or
  INCLUDE = table : "LIKE 'TAB%' "
  ```
- **PARALLEL** – indicates the maximum number of threads of active execution acting on behalf of the export job. For best performance, this parameter should at least be as large as the number of output files specified in the DUMPFILE parameter.
- **FILESIZE** – used to limit the maximum size of the dump file.
- **ATTACH** – this command is used to attach a client session to an existing export job and automatically places the client in an interactive-command interface. A user could also use interactive mode from a terminal other than the one on which the job is run, in which case the expdp system/<password> ATTACH is required.

PERFORMING DATA PUMP IMPORT

This is a utility for loading data from an export dump file created with the expdp utility. The dump file is made up of one or more disk files that contain table data, database object metadata and control information.
You can load data into a target database directly from a source database without any intermediate dump files. This may minimize the total amount of time elapsed time for the entire export and import operation. This is also known as a network import.You can also use the Enterprise Manager to perform a data pump import. While using command line mode you can invoke Data Pump import by executing ***impdp*.**

Example of Table Mode import

```
$ impdp userA/userA  tables=EMPLOYEES  \
  directory=DIR1 \
  dumpfile = EMPLOYEES.dmp \
  content = data_only  \
  logfile = impemployees.dmp
```

Example of Schema Mode Import

```
 $  expdp  system/manager  \
SCHEMA=userA    \
REMAP_SCHEMA= userA:userB  \
DUMPFILE = dir1:sch1%U.dmp, dir2:sch2%U.dmp  \
EXCLUDE=constraint,index  \
TABLE_EXISTS_ACTION = replace \
LOGFILE= impschema.log \
```

Important Parameters

- **CONTENT** – allows you to filter the data and metadata that import loads. A value of DATA_ONLY loads only the table row data. METADATA_ONLY loads only the metadata. ALL loads both.

- **EXCLUDE** – used in a schema mode import. It allows to filter the metadata that is imported by specifying database objects that you want to exclude from the import job. All objects contained within the source, and all their dependent objects are included except those specified in the EXCLUDE statement.

- **TABLE_EXISTS_ACTION** – instructs import about what should be done if a table being imported already exists. You can specify a value of REPLACE drops the existing table and recreates it using export.

- **REMAP_SCHEMA** – used to import a user into another schema. Takes the format of : Original_schema : new_schema.

EXAM OBJECTIVE: CREATE EXTERNAL TABLES FOR DATA POPULATION

The Oracle's External table feature allows a user to create virtual table from a flat-file, which can be queried without having to first load the data into the database.

Table metadata is created in the database, and the virtual table can be accessed using SQL, PL/SQL or Java.

However, these external files are only Read-only. DML operations, INSERT, UPDATE, and DELETE are not allowed on them. Indexes cannot be created on them either. An important benefit of external tables is the ability to pipeline external data directly from the loading into the transformation phase.

Example on an external table

```
CREATE TABLE orders_external
(Order_no     NUMBER,
 Order_desc          VARCHAR2(50),
 ….
Quantity_purchased NUMBER)

ORGANIZATION EXTERNAL
    (TYPE loader_type
    DEFAULT DIRECTORY stage_directory
    ACCESS PARAMETERS
    (< records , fields similar to SQL*LOADER>,
    BADFILE          '/home/oracle/badorders.ext',
    LOGFILE '/home/oracle/logorders.ext'
    LOCATION ('orders1.txt','orders2.txt'))
PARALLEL 3

REJECT LIMIT 200;
```

Created using the CREATE DIRECTORY command.

Badfile and logfile directories should already exist.

Flat files containing data

In Oracle 10g, the external tables can also be written to. It is possible to use the CREATE TABLE AS SELECT command to populate an external that is composed of proprietary format (Direct Path API) flat files that are operating system independent.

Loading data, in the context of external tables is the act of data being read from an external table and loaded into a table in the database. Unloading data is reading data from a table and populating an external table. Loading and Unloading of data is done using the new Data Pump access driver.

The example displayed explains the creation of an external table:

1. Connect as a privileged user and create a directory using a tool like SQL*Plus.

```
SQL> CREATE OR REPLACE DIRECTORY ext_tab_dir
       AS '/home/oracle/externals/';
```

2. Grant read and write privileges on the directory to the user who is loading or unloading the external table.

```
SQL> GRANT READ, WRITE ON DIRECTORY ext_tab_dir
       TO  usera;
```

3. Connect as userA, who wishes to unload data.

```
SQL> CONNECT userA/userA;
```

4. In this command you are unloading data into two flat files, out1.ext and out2.ext using the result of joining two Oracle database tables.

```
SQL> CREATE TABLE out_ext
       (first_name, last_name, depart_name)
       ORGANIZATION EXTERNAL
       (
       TYPE ORACLE_DATAPUMP
       DEFAULT DIRECTORY ext_tab_dir
       LOCATION ('out1.ext','out2.ext')
       )
      PARALLEL
      AS
      SELECT e.first_name, e.last_name, d.department
      FROM employees e  JOIN  departments d
      USING  (department_id)
      WHERE d.department_name IN
      ('MARKETING', 'TRAINING','PRODUCTION');
```

5. Once the external table is creating, the flat files will also be created. To query the external table from within SQL*Plus you can issue:

```
SQL> SELECT * FROM out_ext;
```

6. To view the two files created, you can use as OS command such as

```
$ more out1.ext
$ more out2.ext
```

In the two files created the row data and the metadata are stored in an XML format.

EXAM OBJECTIVE: Define your external table properties

PROJECTED COLUMNS IN EXTERNAL TABLES

In earlier versions of Oracle, prior to 10g, only the columns referenced by the SQL statement were projected out by the access driver. The access driver would parse the input data stream and the external table service that performed data conversions would reject rows due to conversion errors or data format errors. In Oracle 10g, you can get a consistent result set independent of the columns referenced by the SQL statement accessing the external table.

Example of Projected Columns in External tables

1. Consider a flat file called ***abc.dat*** that contains the following data:

```
1234,      3,  3345,  34.33,  150
1235,      5,  3324,  42.35,  200
1236, 2342,  3833,  33.43,  499
```

Let use assume the data should be loaded into an external table that has the second column defined as varchar2(3). The last row could pose a problem.

2. Create an external table as given below:

```
CREATE TABLE abc_table
(Col1       number(12),
 Col2       number(3),
 Col3       number(5),
 Col4       number(5,2),
 Col5       number(4))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
  DEFAULT DIRECTORY ext_tab_dire
  ACCESS PARAMETERS
(RECORDS DELIMITED BY NEWLINE
 FIELDS TERMINED BY ',')
LOCATION ('abc.dat')
)
REJECT LIMIT UNLIMITED;
```

3. Now alter the table to set the PROJECT COLUMN attribute to ALL. This is the default.

```
SQL> ALTER TABLE abc_table PROJECT COLUMN ALL;
SQL> SELECT COUNT( COL1) FROM ABC_TABLE;

  COUNT(*)
         2
```

The loader log file records an error that the third record was rejected when the external table was accessed because the COL2 had more than 3 numbers. What should be pointed out is that the row never reached SQL query processing even though you are referencing the external table using the column with a name of COL1.You can view the abc.bad file to view the row that was rejected.

4. Now set the PROJECT COLUMN attribute to REFERENCED, and run the same query above.

```
SQL> ALTER TABLE abc_table PROJECT COLUMN REFERENCED;
SQL> SELECT COUNT( COL1) FROM ABC_TABLE;

  COUNT(*)
         3
```

The external table is being referenced with the column name COL1 in the select statement. This query projected the external data to the field needed (COL1) so all the three records were accepted.