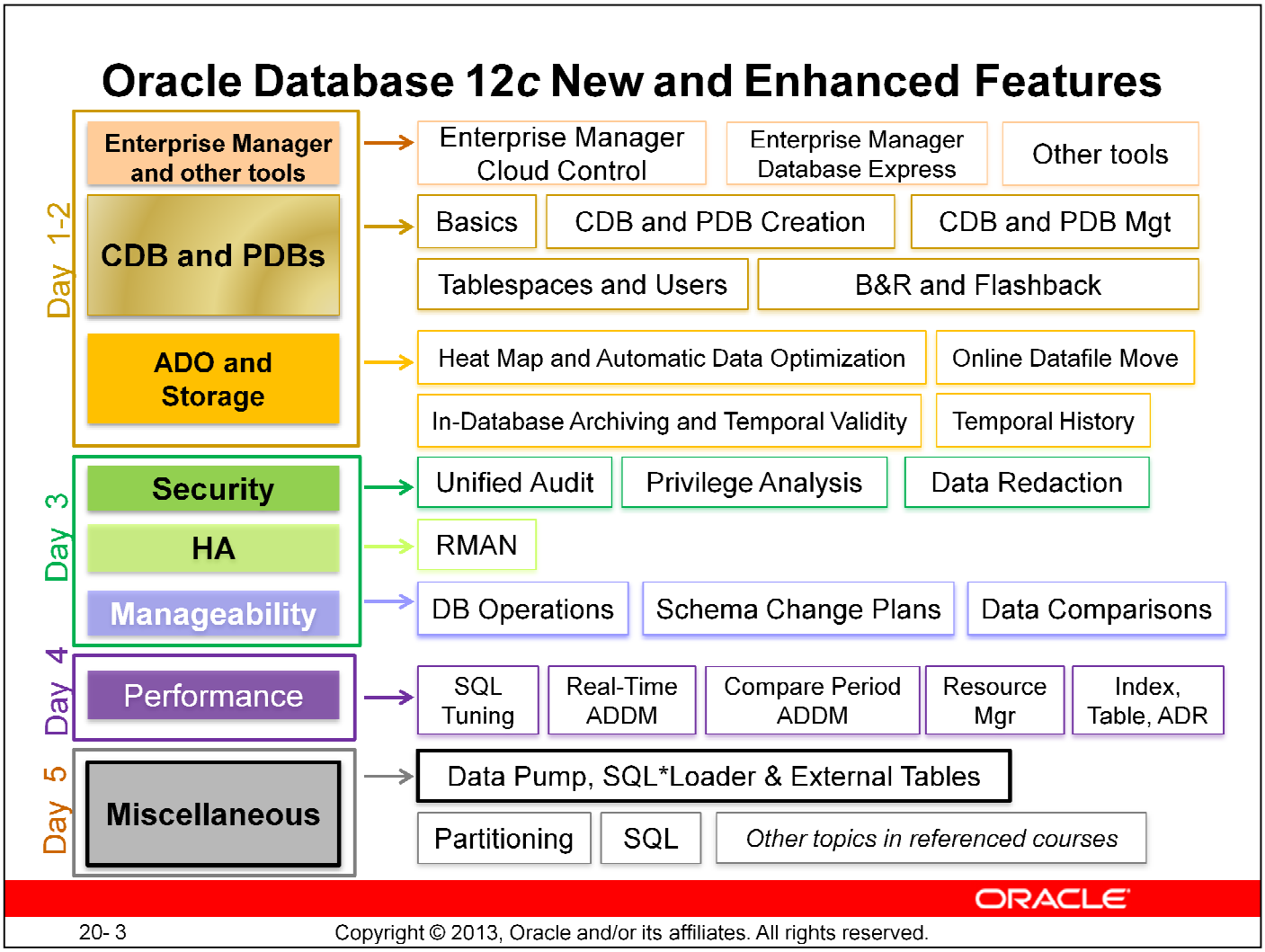
Module

Miscellaneous

20

Oracle Data Pump, SQL*Loader, and External Tables

Schedule:	Timing	Topic
	45 minutes	Lecture
	35 minutes	Practice
	80 minutes	Total



There are several **Oracle Data Pump, SQL*Loader,** and **External Tables** enhancements. You will also examine **SQL*Loader Express Mode** feature.

Objectives

After completing this lesson, you should be able to:

- Describe Oracle Data Pump Full Transportable
- Describe each Oracle Database 12c new feature for Data Pump, SQL*Loader, and External Tables
- Describe applicable use cases and configuration details for the Oracle Data Pump and SQL*Loader new features
- Describe SQL*Loader Express Mode

ORACLE

Note

For a complete understanding of Oracle Data Pump new features and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Utilities 12c Release 1 (12.1)*

12c

Full Transportable Export/Import: Overview

⇨ Upgrading to a new release of Oracle Database
⇨ Transporting a database to a new computer system

- Transport a non-CDB (Oracle Database 11.2.0.3 and up) into another non-CDB.
- Transport a non-CDB (11.2.0.3 and +) into a CDB as a PDB.
- Transport a PDB into another PDB.
- Transport a PDB into a non-CDB.

ORACLE

Full Transportable Export/Import

The full transportable export/import feature is useful in several scenarios:

- **Upgrading to a new release of Oracle Database:** You can use full transportable export/import to upgrade a database from 11.2.0.3 or higher to Oracle Database 12*c*. To do so, install Oracle Database 12*c* and create an empty database. Next, use full transportable export/import to transport the 11.2.0.3 database into the Oracle Database 12*c* database.
- **Transporting a database to a new computer system:** You can use full transportable export/import to move a database from one computer system to another. You might want to move a database to a new computer system to upgrade the hardware or to move the database to a different platform.
- **Transporting a non-CDB into a non-CDB or CDB:** Transported into a CDB, the transported database becomes a PDB associated with the CDB. Full transportable export/import can move an 11.2.0.3 or higher database into an Oracle Database 12*c* database efficiently.

Full Transportable Export/Import: Usage

A full transportable export exports all objects and data necessary to create a complete copy of the database.

- TRANSPORTABLE=ALWAYS parameter
- FULL parameter

```
$ expdp user_name FULL=y DUMPFILE=expdat.dmp
            DIRECTORY=data_pump_dir TRANSPORTABLE=always
            VERSION=12.0 LOGFILE=export.log
```

A full transportable import imports a dump file only if it has been created using the transportable option during export.

- TRANSPORT_DATAFILES
- If NETWORK_LINK used, requires:
  - TRANSPORTABLE=ALWAYS parameter

ORACLE

Full Transportable Export

A full transportable export exports all objects and data necessary to create a complete copy of the database. A mix of data movement methods is used:

- Objects residing in transportable tablespaces have only their metadata unloaded into the dump file set. The data itself is moved when you copy the data files to the target database. The data files that must be copied are listed at the end of the log file for the export operation.
- Objects residing in non-transportable tablespaces (for example, SYSTEM and SYSAUX) have both their metadata and data unloaded into the dump file set, using direct path unload and external tables.

The example shows a full transportable export using the VERSION parameter. The VERSION parameter is required if the source database is an 11.2.0.3 or a higher Oracle Database 11*g* release.

Performing a full transportable export has the following restrictions:

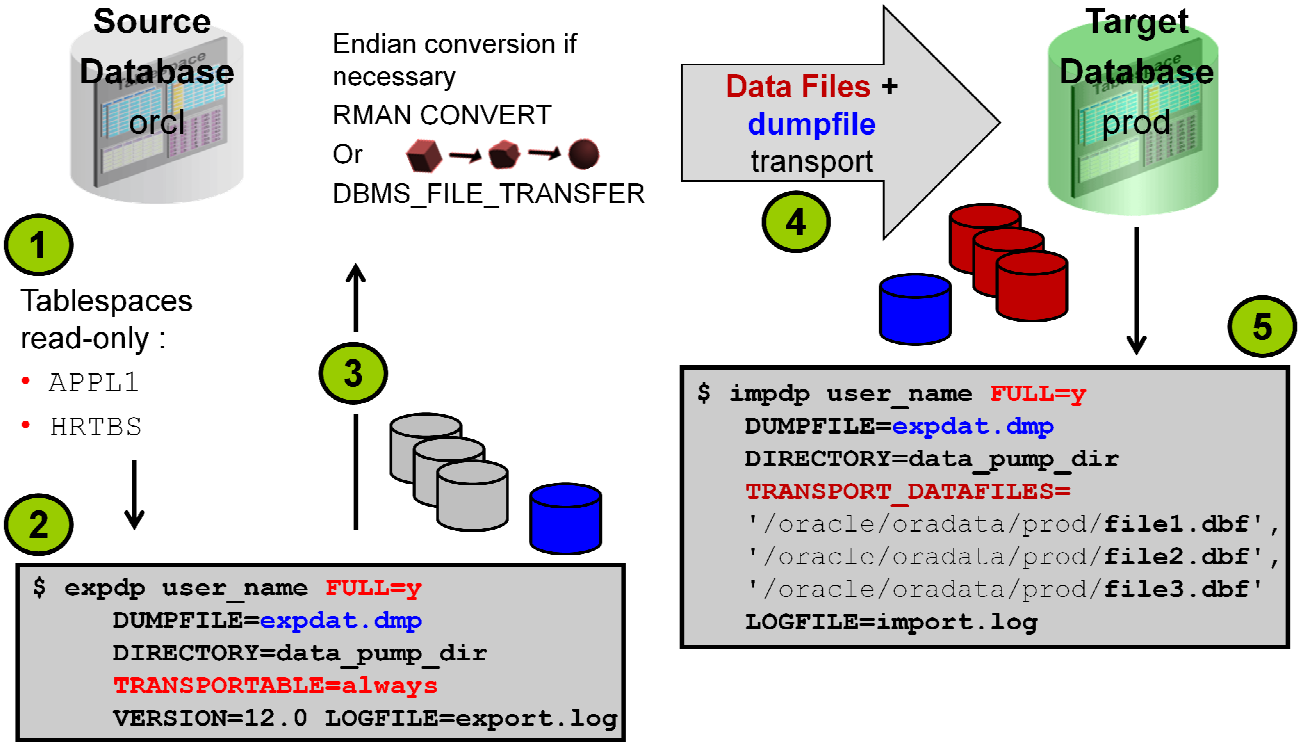
- If the database being exported contains either encrypted tablespaces or tables with encrypted columns (either Transparent Data Encryption (TDE) columns or SecureFile LOB columns), then the ENCRYPTION_PASSWORD parameter must also be supplied.
- The source and target databases must be on platforms with the same endianness if there are encrypted tablespaces in the source database.
- If the source platform and the target platform are of different endianness, you must convert the data being transported so that it is in the format of the target platform. Use either the DBMS_FILE_TRANSFER package or the RMAN CONVERT command.
- A full transportable export is not restartable.
- All objects with storage that are selected for export must have all of their storage segments either entirely within administrative, non-transportable tablespaces (SYSTEM / SYSAUX) or entirely within user-defined, transportable tablespaces. Storage for a single object cannot straddle the two kinds of tablespaces.
- When transporting a database over the network using full transportable export, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable export, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.
- If both the source and target databases are running Oracle Database 12*c* Release 1 (12.1), then to perform a full transportable export, either the Oracle Data Pump VERSION parameter must be set to at least 12.0. or the COMPATIBLE database initialization parameter must be set to at least 12.0 or later.
- Full transportable exports are supported from an 11.2.0.3 source database.

Full Transportable Import

Performing a full transportable import has the following requirements:

- If you are using a network link, then the database specified on the NETWORK_LINK parameter must be Oracle Database 11*g* release 2 (11.2.0.3) or later, and the Oracle Data Pump VERSION parameter must be set to at least 12. (In a non-network import, VERSION=12 is implicitly determined from the dump file.)
- If the source platform and the target platform are of different endianness, then you must convert the data being transported so that it is in the format of the target platform. You can use the DBMS_FILE_TRANSFER package or the RMAN CONVERT command to convert the data.
- A full transportable import of encrypted tablespaces is not supported in network mode or dump file mode if the source and target platforms do not have the same endianess.
- When transporting a database over the network using full transportable import, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable import, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.

Full Transportable Export/Import: Example



To perform a Full Transportable operation, perform the following steps:

1. Before the export, make all of the user-defined tablespaces in the database read-only.
2. Invoke the Oracle Data Pump export utility as a user with DATAPUMP_EXP_FULL_DATABASE role and specify the full transportable export options: FULL=Y, TRANSPORTABLE=ALWAYS.
   The LOGFILE parameter is important because it will contain the list of data files that need to be transported for the import operation.
   To perform the operation on an 11.2.0.3 database, use the VERSION parameter. Full transportable import is supported only for Oracle Database 12*c* databases.
3. Before the import, transport the dump file.
4. Transport the data files that you may have converted. If you are transporting the database to a platform different from the source platform, then determine if cross-platform database transport is supported for both the source and target platforms. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of each tablespace in the database either at the source or target database using either DBMS_FILE_TRANSFER or RMAN CONVERT command.
5. Invoke the Oracle Data Pump import utility as a user with DATAPUMP_IMP_FULL_DATABASE role and specify the full transportable import options: FULL=Y, TRANSPORT_DATAFILES.
6. Make the source tablespaces read-write. You can perform this operation before step 5.

Transporting a Database Over the Network: Example

Transport a database over the network: perform an import using the NETWORK_LINK parameter.

1. Create a database link in the target to the source database.
2. Make the user-defined tablespaces in the source database read-only.
3. Transport the datafiles for all of the user-defined tablespaces from the source to the target location.
4. Perform conversion of the datafiles if necessary.
5. Import in the target database.

```
$ impdp username full=Y network_link=sourcedb
      transportable=always
      transport_datafiles='/oracle/oradata/prod/sales01.dbf',
                          '/oracle/oradata/prod/cust01.dbf'
    encryption_password=pass1 version=12 logfile=import.log
```

ORACLE

To transport a database over the network, use import with the NETWORK_LINK parameter. The import is performed using a database link, and there is no dump file involved.

- If the source database is an 11.2.0.3 database or a higher Oracle Database 11*g* release database, then the VERSION parameter is required and must be set to 12. If the source database is an Oracle Database 12*c* database, then the VERSION parameter is not required.
- If the source database contains any encrypted tablespaces or tablespaces containing tables with encrypted columns, then you must specify the ENCRYPTION_PASSWORD parameter.
- The Oracle Data Pump network import copies the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as SYSTEM and SYSAUX.
- When the import is complete, the user-defined tablespaces are in read-write mode.
- Make the user-defined tablespaces read-write again at the source database.

12c

Disabling Logging for Oracle Data Pump Import: Overview

- The goal is to deliver faster data loads:
  - Redo log information is not written to disk.
  - Greater I/O bandwidth is dedicated to loading.
- This option is particularly useful for:
  - Performing large data loads
  - Populating new databases
- A full database backup is recommended after the data load finishes.

ORACLE

Oracle Data Pump provides an option to disable logging during an import operation. The result is quicker imports, because redo log information is not written to disk or archived, allowing greater I/O bandwidth to be dedicated to data loading. Disabling logging is particularly useful for large data loads or jobs that initially populate a new database.

If you disable logging, Oracle recommends that you perform a full Recovery Manager (RMAN) backup after the Oracle Data Pump import operation is completed. Also, in the unlikely event of media failure in the middle of the import, you must restart the import.

Disabling Logging for Oracle Data Pump Import: Usage

- Command-line syntax:

```
$ impdp ... TRANSFORM=DISABLE_ARCHIVE_LOGGING:Y|N[:TABLE|INDEX]
```

- Notes:
  - Logging is not completely eliminated. A small amount of log activity is still generated by the import.
  - Logging is not disabled if the database is in `FORCE LOGGING` mode.

ORACLE

You can disable logging using a new metadata transform parameter, `DISABLE_ARCHIVE_LOGGING`. An outline of the syntax is shown in the slide and some examples follow on the next page.

If set to `y`, then the logging attributes for the specified object types, `TABLE` and/or `INDEX` are disabled before the data is imported. If set to `n` (the default), then archive logging is not disabled during import. After the data has been loaded, the logging attributes for the objects are restored to their original settings. If no object type is specified, then the `DISABLE_ARCHIVE_LOGGING` behavior is applied to both `TABLE` and `INDEX` object types.

Note that the operations against the master table that is used by Oracle Data Pump to coordinate its activities are logged, even if logging is disabled for the Oracle Data Pump import session.

If the database is in `FORCE LOGGING` mode, logging is not disabled during the Oracle Data Pump load even if the option to do so is specified.

Disabling Logging for Oracle Data Pump Import: Command-Line Examples

```
$ impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp SCHEMAS=hr
  TRANSFORM=DISABLE_ARCHIVE_LOGGING:Y
```
1

```
$ impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp SCHEMAS=hr
  TRANSFORM=DISABLE_ARCHIVE_LOGGING:Y:INDEX
```
2

```
$ impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp SCHEMAS=hr
  TRANSFORM=DISABLE_ARCHIVE_LOGGING:Y
  TRANSFORM=DISABLE_ARCHIVE_LOGGING:N:TABLE
```
3

The slide shows three `impdp` command-line examples where logging is disabled.

- Example 1 is a simple case, where logging is disabled for both `TABLE` and `INDEX` object types.
- Example 2 disables logging is disabled for `INDEX` only.
- Example 3 shows a different way to achieve the same outcome as example 2.

12c

Exporting Views as Tables: Overview

- The following are exported:
  - A table definition and the view data (instead of just the view definition SQL)
  - Dependent objects (such as constraints and grants)
- This option is particularly useful for:
  - Exporting a data subset
  - Exporting a denormalized data set spanning multiple tables

ORACLE

Oracle Data Pump provides the option to export a view as a table. Instead of just exporting the view definition SQL, Oracle Data Pump writes into the dump file a corresponding table definition and all the data visible in the view.

At import time, Oracle Data Pump can create a table with the same geometry (columns and data types) as the original view and populate it with the exported data.

Objects depending on the view are also exported as if they were defined on the table. For example, grants that are associated with the view are recorded as grants on the corresponding table in the export dump file. Dependent objects that do not apply to tables are not exported.

The ability to export views as tables can be used to export data subsets for development and testing, data replication, and offline archival purposes.

You can also use this feature to export a denormalized data set spanning multiple related tables. This kind of operation is commonly encountered when data is extracted from transactional systems and loaded into decision support systems and data warehouses.

Exporting Views as Tables: Usage

- Command-line syntax:

```
$ expdp ... VIEWS_AS_TABLES=[schema_name.]view_name ...
```

  - Only relational views with scalar columns are supported.
  - Exported data is unencrypted by default.

You can export views as tables by using the command-line parameter `VIEWS_AS_TABLES`. You can also set it programmatically in PL/SQL by using the `DBMS_DATAPUMP.METADATA_FILTER` procedure. In both cases, you must provide a view name, which can be schema-qualified.

Exporting views as tables is limited only to relational views with scalar columns. More complex views, including views that reference object types or functions, are not supported.

The `VIEWS_AS_TABLES` parameter unloads view data in unencrypted format. If you unload sensitive data, Oracle strongly recommends that you enable encryption on the export operation and that you ensure that the imported table is encrypted.

Exporting Views as Tables: Command-Line Examples

```
$ expdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=hr1.dmp
  VIEWS_AS_TABLES=employees_v TABLES=departments
```
1

```
$ expdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=hr2.dmp
  VIEWS_AS_TABLES=managers_v,oe.orders_v
```
2

```
$ impdp hr1/hr1 DIRECTORY=dpump_dir1 DUMPFILE=hr2.dmp
  VIEWS_AS_TABLES=managers_v REMAP_TABLE=managers_v:managers
```
3

```
$ impdp hr1/hr1 VIEWS_AS_TABLES=employees_v NETWORK_LINK=hr_link
  REMAP_TABLE=employees_v:employees TABLE_EXISTS_ACTION=append
```
4

ORACLE

The slide shows five examples using the VIEWS_AS_TABLES command-line parameter.

- Example 1 is a simple case, where the VIEWS_AS_TABLES parameter specifies a single view (hr.employees_v). You can use this parameter by itself or with the TABLES parameter. If you use either parameter, Oracle Data Pump performs a table-mode export.
- Example 2 exports two views as tables (hr.managers_v and oe.orders_v).
- Example 3 uses the export dump file that was created in example 2. It demonstrates using the VIEWS_AS_TABLES parameter in conjunction with impdp to selectively import managers_v as the managers table.
- Example 4 shows the VIEWS_AS_TABLES parameter being used in conjunction with a network import. In this mode, the syntax for the VIEWS_AS_TABLES parameter is the same as that supported by Oracle Data Pump export (expdp). In this example, the view employees_v is sourced from the database specified by the hr_link database link . The data from employees_v@hr_link is imported into the employees table in the hr1 schema on the target database. Because TABLE_EXISTS_ACTION=append is specified, the data from the source view rows are appended to the hr1.employees table if it already exists.

12c

Specifying the Encryption Password

- New option prompts users to specify encryption password:

```
$ expdp ... ENCRYPTION_PWD_PROMPT=Y ...

...

Password: <Specify password here. Input not echoed>

...
```

- With this feature, encryption password security is enhanced because it is supplied silently at run time.
  - The password is not visible by using commands like `ps`.
  - The password is not stored in scripts.
- Concurrent use of the `ENCRYPTION_PASSWORD` parameter and `ENCRYPTION_PWD_PROMPT=Y` is prohibited and generates an error.

ORACLE

Earlier versions of the Oracle Data Pump utilities (`expdp` and `impdp`) provided support for encrypting data written to the export dump file. Although you can use an Oracle Wallet for transparent encryption, password-based encryption is also available.

To support password-based encryption and enable a password to be specified, a command -line parameter, `ENCRYPTION_PASSWORD`, was added to the Oracle Data Pump utilities. Before Oracle Database 12*c*, using the `ENCRYPTION_PASSWORD` parameter was the only way to set a password for password-based encryption in conjunction with Oracle Data Pump export. The issue with this mechanism is that the encryption password can be easily compromised. Non-privileged system users can typically use utilities such as `ps` on Linux and UNIX to view command -line parameters, including the encryption password. Also, storing the encryption password inside scripts is generally considered to be poor practice from a security perspective.

With Oracle Database 12*c*, the Oracle Data Pump utilities have the ability to prompt the user for an encryption password without the value being echoed as it is entered. The new command-line parameter, `ENCRYPTION_PWD_PROMPT`, can be set to `Y` to instruct the utilities to prompt the user for the encryption password. The password is read from the standard input stream, and terminal echo is suppressed while the password is being entered.

12c

Compressing Tables During Import

- Compression method can now be specified at import time.
  - Independent of compression settings present at export time.
- Command-line syntax and example:

```
$ impdp ... TRANSFORM=TABLE_COMPRESSION_CLAUSE:NONE|"<comp-clause>" ...
```

```
$ impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=export.dmp
  TRANSFORM=TABLE_COMPRESSION_CLAUSE:"COMPRESS FOR OLTP"
```

- Syntax notes:
  - `NONE` => compression clause omitted from `CREATE TABLE`
  - `<comp-clause>` => valid table compression clause

ORACLE

Using earlier versions of Oracle Data Pump, tables are imported with the same compression settings that were present during export. Oracle Database 12*c* enables Oracle Data Pump to specify the compression method at import time regardless of the settings in the source database.

The option to compress (or uncompress) tables during import is provided by a new metadata transform parameter, `TABLE_COMPRESSION_CLAUSE`, which can be set on the `impdp` command line or programmatically by using the `DBMS_DATAPUMP.METADATA_TRANSFORM` PL/SQL procedure.

If the parameter is set to `NONE`, the table compression clause is omitted and imported tables get the default compression setting associated with the tablespace. This option is particularly useful when you want to apply a uniform compression method to a series of tables that may have had different compression settings in the source database.

Otherwise, the parameter value must be a valid table compression clause, such as `NOCOMPRESS`, `COMPRESS FOR OLTP`, and `COMPRESS FOR QUERY LOW`.

Table compression clauses containing more than one keyword must be enclosed in double quotation marks when they are specified on the `impdp` command line. Because quotation marks in the command line can have special meaning in some operating systems, consider using a parameter file to set the transform parameter. Also, remember that `COMPRESS FOR QUERY` and `COMPRESS FOR ARCHIVE` are valid only in conjunction with Exadata storage.

12c

Creating SecureFile LOBs During Import

- Option to specify LOB storage method at import time
- Command-line syntax and example:

```
$ impdp ... TRANSFORM=LOB_STORAGE:SECUREFILE|BASICFILE|DEFAULT|NO_CHANGE
  ...
```

```
$ impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=export.dmp
  LOB_STORAGE:SECUREFILE
```

- Syntax notes:
  - DEFAULT => no lob storage clause for CREATE TABLE
  - NO_CHANGE => settings specified in dump file

ORACLE

Oracle Database 12*c* enables Oracle Data Pump to specify the LOB storage method at import time regardless of the settings in the source database. This capability provides a straightforward mechanism for users to migrate from BasicFile LOBs to SecureFile LOBs.

The option to specify the LOB storage method during import is provided by a new metadata transform parameter, LOB_STORAGE, which can be set on the impdp command line or programmatically by using the DBMS_DATAPUMP.METADATA_TRANSFORM PL/SQL procedure.

By setting the parameter to SECUREFILE or BASICFILE, you specify the corresponding LOB storage method for all LOBs that are associated with the import session. If you set the parameter to DEFAULT, the LOB storage clause is omitted and imported tables get the default setting that is associated with the session or instance. If you set the parameter to NO_CHANGE, the LOB storage clauses that are specified in the export dump file are used.

Quiz

Setting `TRANSFORM=TABLE_COMPRESSION_CLAUSE:NONE` on the impdp command causes all the tables created by the impdp session to be uncompressed.

a. True

b. False

Answer: b

Imported tables get the default compression setting associated with the tablespace.

12c

SQL*Loader Support for Direct-Path Loading of Identity Columns

- An identity column is a numeric column that is assigned an integer value from a sequence generator for each INSERT performed on the table.
- SQL*Loader supports direct-path loading of identity columns:

Column specification	Action performed by SQL*Loader
col1 NUMBER GENERATED ALWAYS AS IDENTITY	col1 cannot be loaded explicitly. The column value is always provided by the sequence generator.
col2 NUMBER GENERATED BY DEFAULT AS IDENTITY	col2 can be loaded explicitly. If it is not, the column value is provided by the sequence generator. Explicitly loaded NULL values cause the corresponding row to be rejected because identity columns implicitly carry the NOT NULL constraint.
col3 NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY	col3 can be loaded explicitly. If it is not, the column value is provided by the sequence generator. Explicitly loaded NULL values are replaced by sequence values.

ORACLE

SQL*Loader new features start with support for direct-path loading of identity columns.

An identity column is a numeric column that is assigned an increasing or decreasing integer value from a sequence generator for each INSERT statement that is performed on the associated table. You specify identity columns by using the GENERATED ... AS IDENTITY clause in association with a CREATE TABLE or ALTER TABLE statement. Identity columns are new in Oracle Database 12*c*.

In conjunction with this new feature, SQL*Loader supports direct-path loading of identity columns. The table on the slide describes the actions that are performed by SQL*Loader for different types of identity columns. These actions occur automatically and are consistent with the normal behavior of identity columns.

12c

SQL*Loader and External Table Enhancements

Wildcards allowed in names of data files

- `INFILE('emp*.dat')` or `('emp?.dat')`

Support for CSV files with embedded newlines

- `FIELDS CSV WITH EMBEDDED …`

Table-level `NULLIF` and Date formats can be specified once for all character and date fields

- `FIELDS DATE FORMAT "DD-Month-YYYY"`
- `FIELDS TERMINATED BY "," NULLIF = "NA"`

With `FIELD NAMES` clause, the first record in the data file contains the names and order of the fields

- `FIELD NAMES FIRST FILE`

Specific to external tables: `ALL FIELDS OVERRIDE`

- Default attributes for columns so you only have to specify columns that do not match the default

ORACLE

The following statements can be applied to SQL*Loader and External Tables. The syntax may differ when using SQL*Loader or External Tables.

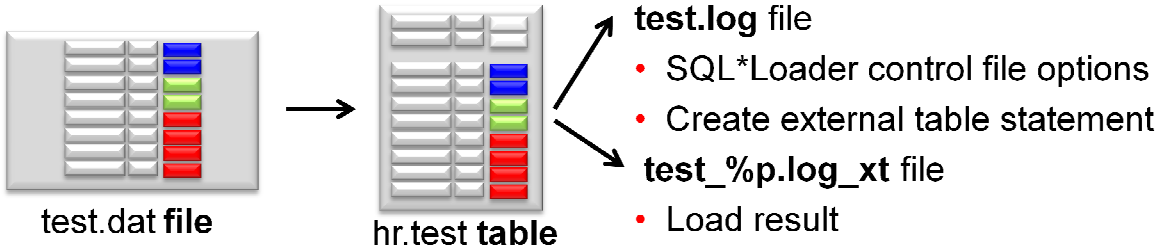
- Wildcards are allowed in data file names: "*" is for multiple characters and "?" is for single characters.
- CSV files are supported in Oracle Database 12*c* under conditions. Read *Oracle Database Utilities 12c Release 1 (12.1)* to get the conditions under which CSV files data can be loaded.
- Table-level `NULLIF` and date format can be specified once for all fields and can be overridden for an individual field. It applies to all character and date fields for `NULLIF` and to all date fields for date format.
- The first record containing the names and order of the fields can be in the first file or every file.
- `FIRST FILE` indicates that the first data file contains a list of field names for the data in the first record. `ALL FILES` indicates that all data files contain a list of field names for the data in the first record. The first record is skipped in each data file when the data is processed. The fields can be in a different order in each data file.
- The last statement applies only to external tables.The `ALL FIELDS OVERRIDE` clause tells the access driver that all fields are present and that they are in the same order as the columns in the external table. You only need to specify fields that have a special definition.

12c

SQL*Loader Express Mode

- It is simple to use.
- Specify a table name to initiate an Express Mode load:
  - The table columns must be scalar datatypes (character, number, or datetime).
  - The data file contains only delimited character data.
  - SQL*Loader uses table column definitions to determine input data types.
- There is no need to create a control file.

```
$ sqlldr hr TABLE=test
```



If you activate SQL*Loader Express Mode, specifying only the username and the TABLE parameter, it uses default settings for a number of other parameters. You can override most of the defaults by specifying additional parameters on the command line.

SQL*Loader express mode generates two files. The names of the log files come from the name of the table (by default).

- A log file that includes:
  - The control file output
  - A SQL script for creating the external table and performing the load using a SQL `INSERT AS SELECT` statement.

  Neither the control file nor the SQL script are used by SQL*Loader express mode. They are made available to you in case you want to use them as a starting point to perform operations using regular SQL*Loader or stand-alone external tables.
- A log file similar to a SQL*Loader log file that describes the result of the operation The ″%p″ represents the process id of the SQL*Loader process.

An example of the input data file named `test.dat` containing two records to load and excerpts of the two log files generated after the load completed. The table `HR.TEST` was created with three columns, `C1 NUMBER`, `C2 VARCHAR2(10)`, and `C3 VARCHAR2(10)`.

```
$ more test.dat
3 C WWW
4 D UUU
$ sqlldr hr TABLE=test
$ more test.log
…
Express Mode Load, Table: TEST
Data File:      test.dat
  Bad File:     test_%p.bad    …
Table TEST, loaded from every logical record.
Insert option in effect for this table: APPEND
Column Name          Position   Len   Term Encl Datatype
-------------------- ---------- ----- ---- ---- ---------
C1                   FIRST               *    , CHARACTER
C2                   NEXT                *    , CHARACTER
C3                   NEXT                *    , CHARACTER
Generated control file for possible reuse:
OPTIONS(EXTERNAL_TABLE=EXECUTE, TRIM=LRTRIM)
LOAD DATA INFILE 'test'  APPEND INTO TABLE TEST
FIELDS TERMINATED BY ","  (  C1,  C2,  C3)
End of generated control file for possible reuse.
…
creating external table "SYS_SQLLDR_X_EXT_TEST"
CREATE TABLE "SYS_SQLLDR_X_EXT_TEST"
("C1" NUMBER,"C2" CHAR(1),"C3" VARCHAR2(20)) ORGANIZATION external(
 TYPE oracle_loader DEFAULT DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000
 ACCESS PARAMETERS
 ( RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
   BADFILE 'SYS_SQLLDR_XT_TMPDIR_00000':'test_%p.bad'
   LOGFILE 'test_%p.log_xt'  READSIZE 1048576
   FIELDS TERMINATED BY "," LRTRIM REJECT ROWS WITH ALL NULL FIELDS
("C1" CHAR(255),"C2" CHAR(255),"C3" CHAR(255)))
  location ('test.dat')) REJECT LIMIT UNLIMITED
executing INSERT statement to load database table TEST
INSERT /*+ append parallel(auto) */ INTO TEST (C1, C2, C3)
SELECT   "C1",  "C2",  "C3" FROM "SYS_SQLLDR_X_EXT_TEST"  …
Table TEST:   2 Rows successfully loaded.
```

Summary

In this lesson, you should have learned how to:

- Describe Oracle Data Pump Full Transportable
- Describe each Oracle Database 12*c* new feature for Oracle Data Pump, SQL*Loader, and External Tables
- Describe applicable use cases and configuration details for the Oracle Data Pump and SQL*Loader new features
- Describe SQL*Loader Express Mode

Practice 20 Overview: Oracle Data Pump Enhancements

These practices cover the following topics:

- Transporting a database by using the Full Transportable feature
- Loading data by using SQL*Loader Expres Mode *(optional)*