19

Moving Data

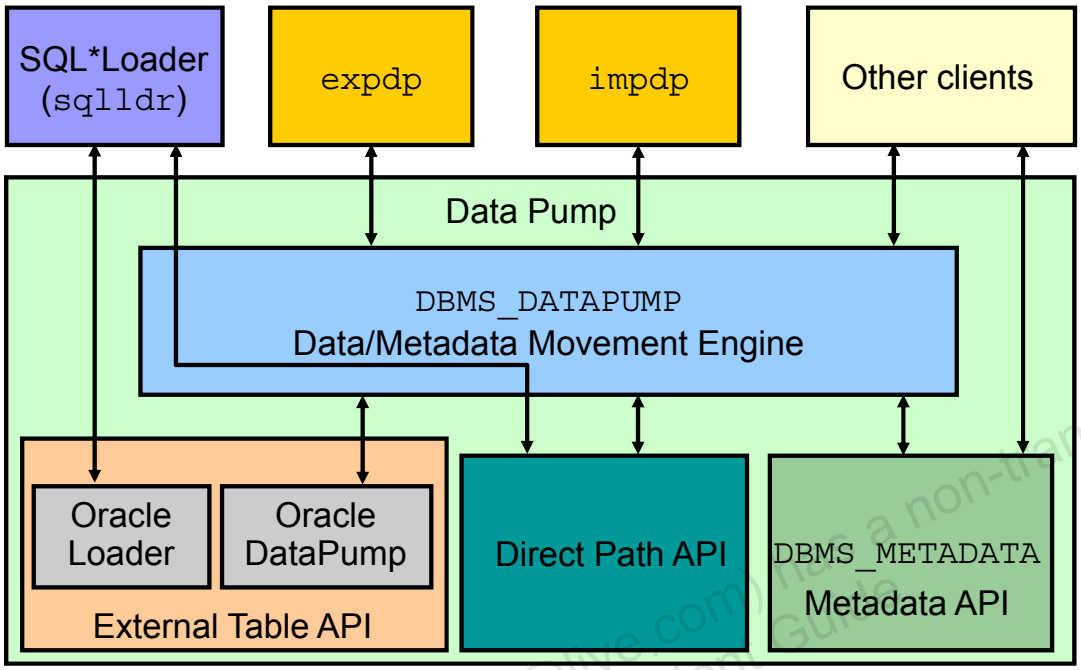
ORACLE

Objectives

After completing this lesson, you should be able to:

- Describe ways to move data
- Explain the general architecture of Oracle Data Pump
- Create and use directory objects
- Use Data Pump Export and Import to move data between Oracle databases
- Use SQL*Loader to load data from a non-Oracle database (or user files)
- Use external tables to move data via platform-independent files

Moving Data: General Architecture



Major functional components:

- **DBMS_DATAPUMP:** Contains the API for high-speed export and import utilities for bulk data and metadata movement
- **Direct Path API (DPAPI):** Oracle Database supports a Direct Path API interface that minimizes data conversion and parsing at both unload and load time.
- **DBMS_METADATA:** Used by worker processes for all metadata unloading and loading. Database object definitions are stored using XML rather than SQL.
- **External Table API:** With the ORACLE_DATAPUMP and ORACLE_LOADER access drivers, you can store data in external tables (that is, in platform-independent files). The SELECT statement reads external tables as though they were stored in an Oracle database.
- **SQL*Loader:** Has been integrated with external tables, providing automatic migration of loader control files to external table access parameters
- **expdp** and **impdp:** Thin layers that make calls to the DBMS_DATAPUMP package to initiate and monitor Data Pump operations
- **Other clients:** Applications (such as replication, transportable tablespaces, and user applications) that benefit from this infrastructure. SQL*Plus may also be used as a client of DBMS_DATAPUMP for simple status queries against ongoing operations.

Oracle Data Pump: Overview

As a server-based facility for high-speed data and metadata movement, Oracle Data Pump:

- Is callable via `DBMS_DATAPUMP`
- Provides the following tools:
  - `expdp`
  - `impdp`
  - GUI interface in Enterprise Manager Cloud Control
- Provides four data movement methods:
  - Data file copying
  - Direct path
  - External tables
  - Network link support
- Detaches from and re-attaches to long-running jobs
- Restarts Data Pump jobs

Oracle Data Pump enables very high-speed data and metadata loading and unloading of Oracle databases. The Data Pump infrastructure is callable via the `DBMS_DATAPUMP` PL/SQL package. Thus, custom data movement utilities can be built by using Data Pump.

Oracle Database provides the following tools:

- Command-line export and import clients called `expdp` and `impdp`, respectively
- Export and import interface in Enterprise Manager Cloud Control

Data Pump automatically decides the data access methods to use; these can be either direct path or external tables. Data Pump uses direct path load and unload when a table's structure allows it and when maximum single-stream performance is desired. However, if there are clustered tables, referential integrity constraints, encrypted columns, or several other items, Data Pump uses external tables rather than direct path to move the data.

The ability to detach from and re-attach to long-running jobs without affecting the job itself enables you to monitor jobs from multiple locations while they are running. All stopped Data Pump jobs can be restarted without loss of data as long as the metainformation remains undisturbed. It does not matter whether the job is stopped voluntarily or involuntarily due to a crash.

Oracle Data Pump: Benefits

Data Pump offers many benefits and many features, such as:

- Fine-grained object and data selection
- Explicit specification of database version
- Parallel execution
- Estimation of export job space consumption
- Network mode in a distributed environment
- Remapping capabilities
- Data sampling and metadata compression
- Compression of data during a Data Pump export
- Security through encryption
- Ability to export XMLType data as CLOBs
- Legacy mode to support old import and export files

The `EXCLUDE`, `INCLUDE`, and `CONTENT` parameters are used for fine-grained object and data selection.

You can specify the database version for objects to be moved (using the `VERSION` parameter) to create a dump file set that is compatible with a previous release of Oracle Database that supports Data Pump.

You can use the `PARALLEL` parameter to specify the maximum number of threads of active execution servers operating on behalf of the export job.

You can estimate how much space an export job would consume (without actually performing the export) by using the `ESTIMATE_ONLY` parameter.

Network mode enables you to export from a remote database directly to a dump file set. This can be done by using a database link to the source system.

During import, you can change the target data file names, schemas, and tablespaces.

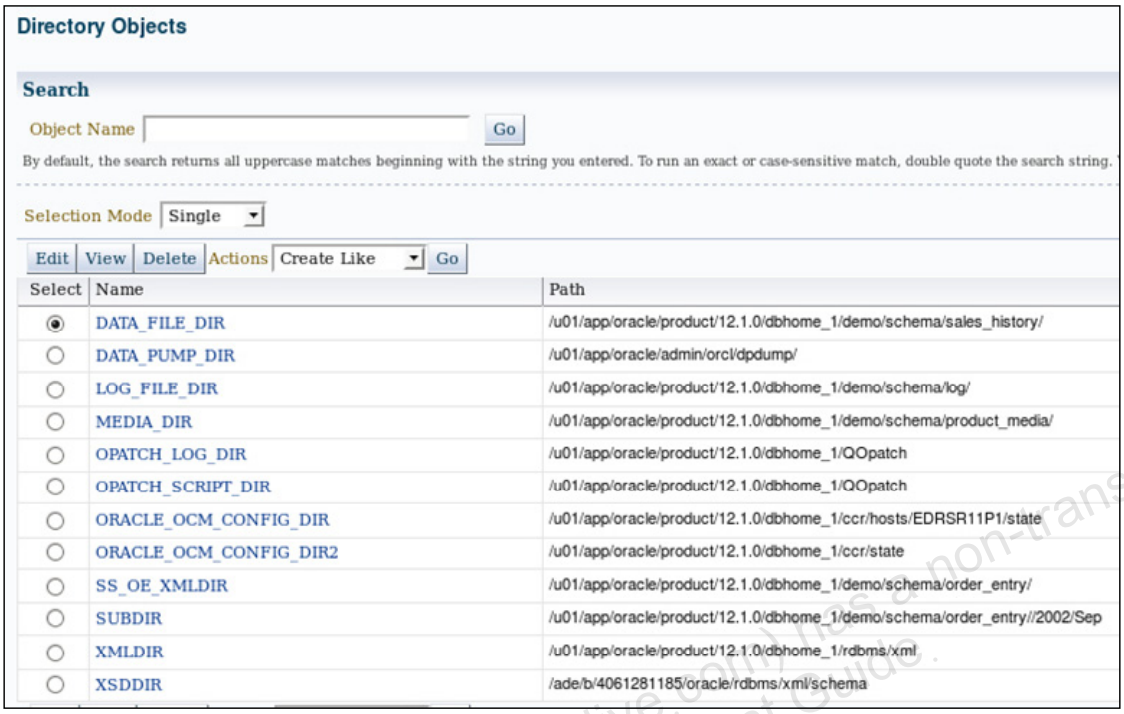
In addition you can specify a percentage of data to be sampled and unloaded from the source database when performing a Data Pump export. This can be done by specifying the `SAMPLE` parameter.

You can use the `COMPRESSION` parameter to indicate whether the metadata should be compressed in the export dump file so that it consumes less disk space. If you compress the metadata, it is automatically uncompressed during import.

Features of Data Pump enable you to:

- Compress both data and metadata, only data, only metadata, or no data during an export
- Specify additional encryption options in the following areas:
  - You can choose to encrypt both data and metadata, only data, only metadata, no data, or only encrypted columns during an export.
  - You can specify a particular encryption algorithm to use during an export.
  - You can specify the type of security to use for performing encryption and decryption during an export. For example, perhaps the dump file set will be imported into a different or remote database and it must remain secure in transit. Or perhaps the dump file set will be imported on-site using the Oracle Encryption Wallet but it may also need to be imported off-site where the Oracle Encryption Wallet is not available.
- Perform table mode exports and imports using the transportable method; specify how partitioned tables should be handled during import operations
- Overwrite existing dump files during an export operation
- Rename tables during an import operation
- Specify that a data load should proceed even if nondeferred constraint violations are encountered (This is valid only for import operations that use the external tables access method.)
- Specify that XMLType columns are to be exported in uncompressed CLOB format regardless of the XMLType storage format that was defined for them
- During an export, specify a remap function that takes as a source the original value of the designated column and returns a remapped value that will replace the original value in the dump file
- Remap data as it is being imported into a new database
- Use legacy mode to support the use of original Export (`exp`) and Import (`imp`) scripts

Directory Objects for Data Pump



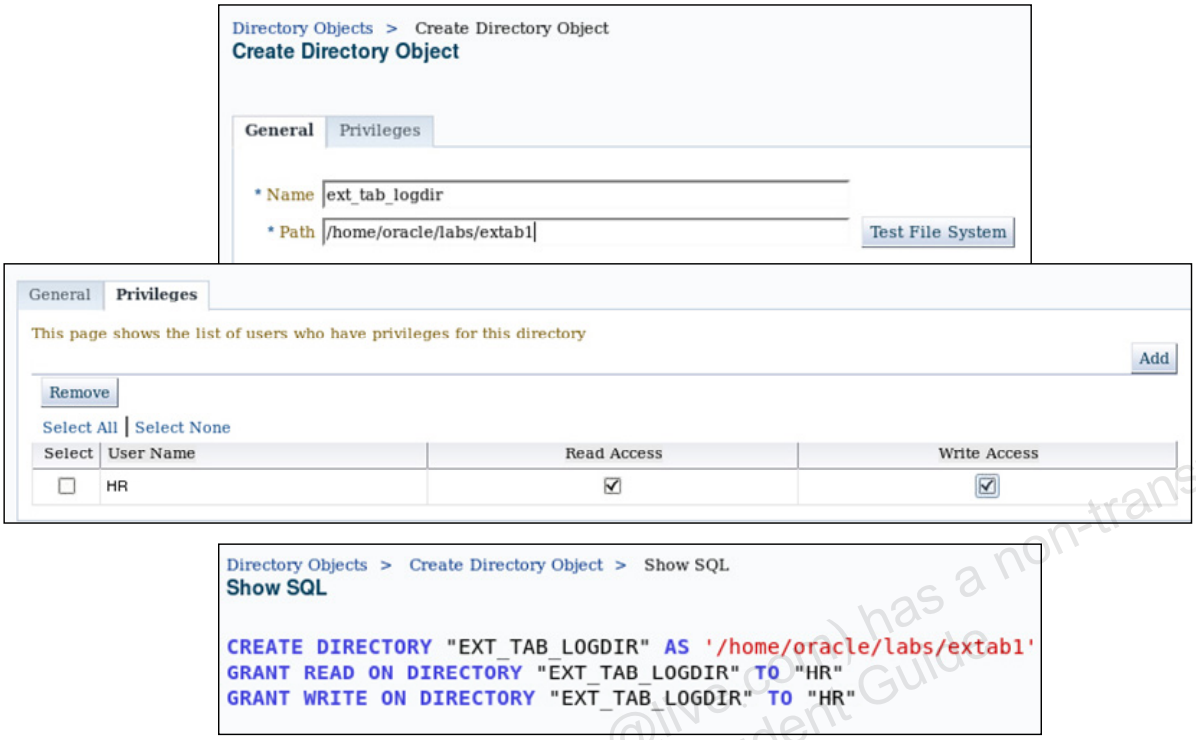
Directory objects are logical structures that represent a physical directory on the server's file system. They contain the location of a specific operating system directory. This directory object name can be used in Enterprise Manager so that you do not need to hard-code directory path specifications. This provides greater flexibility for file management. Directory objects are owned by the `SYS` user. Directory names are unique across the database because all the directories are located in a single name space (that is, `SYS`).

Directory objects are required when you specify file locations for Data Pump because it accesses files on the server rather than on the client.

In Enterprise Manager Cloud Control, select Schema > Database Objects > Directory Objects.

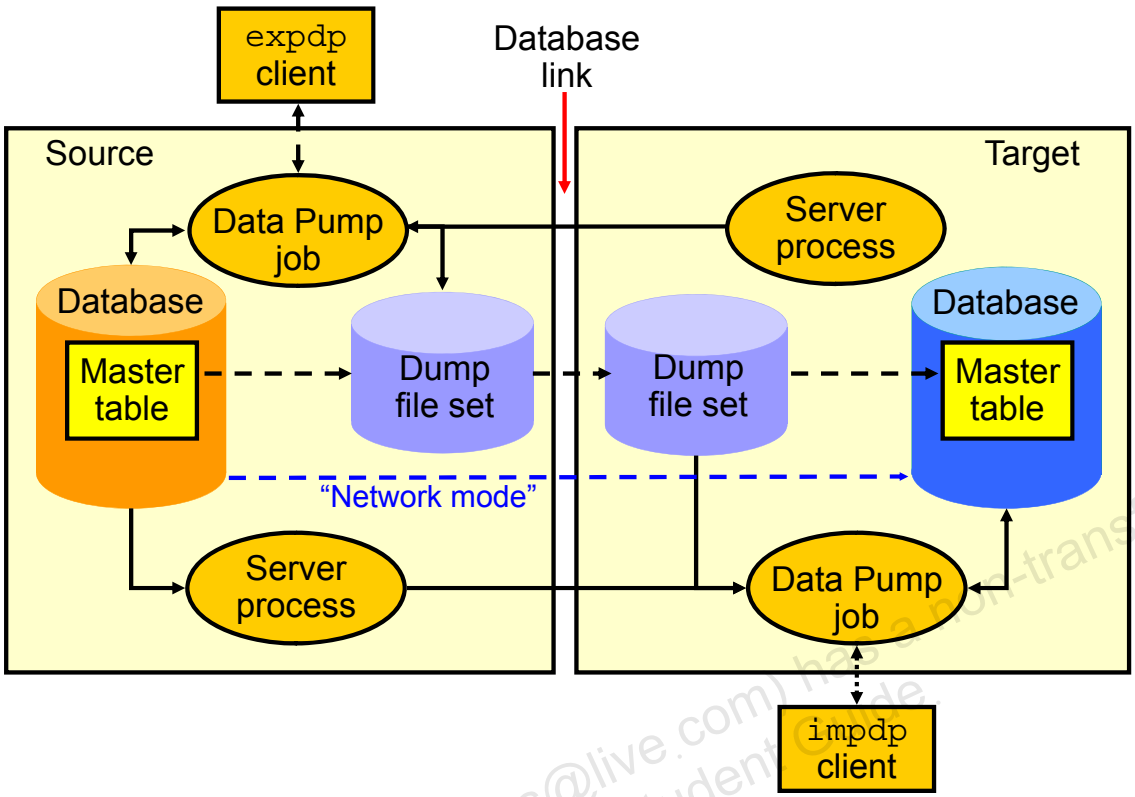
To edit or delete a directory object, select the object and click the appropriate button.

Creating Directory Objects



1. On the Directory Objects page, click Create.
2. Enter the name of the directory object and the OS path to which it maps. OS directories should be created before they are used. You can test this by clicking Test File System. For the test, provide the host login credentials (the OS user who has privileges on this OS directory).
3. Permissions for directory objects are not the same as OS permissions on the physical directory on the server file system. You can manage user privileges on individual directory objects. This increases the level of security and gives you granular control over these objects. On the Privileges page, click Add to select the user to which you give read or write privileges (or both).
4. Click Show SQL to view the underlying statements. Click Return when finished.
5. Click OK to create the object.

Data Pump Export and Import Clients: Overview



Data Pump Export is a utility for unloading data and metadata into a set of operating system files called *dump file sets*. Data Pump Import is used to load metadata and data stored in an export dump file set into a target system.

The Data Pump API accesses its files on the server rather than on the client.

These utilities can also be used to export from a remote database directly to a dump file set, or to load the target database directly from a source database with no intervening files. This is known as *network mode*. This mode is particularly useful to export data from a read-only source database.

At the center of every Data Pump operation is the master table (MT), which is a table created in the schema of the user running the Data Pump job. The MT maintains all aspects of the job. The MT is built during a file-based export job and is written to the dump file set as the last step. Conversely, loading the MT into the current user's schema is the first step of a file-based import operation and is used to sequence the creation of all objects imported.

Note: The MT is the key to Data Pump's restart capability in the event of a planned or unplanned stopping of the job. The MT is dropped when the Data Pump job finishes normally.

Data Pump Utility: Interfaces and Modes

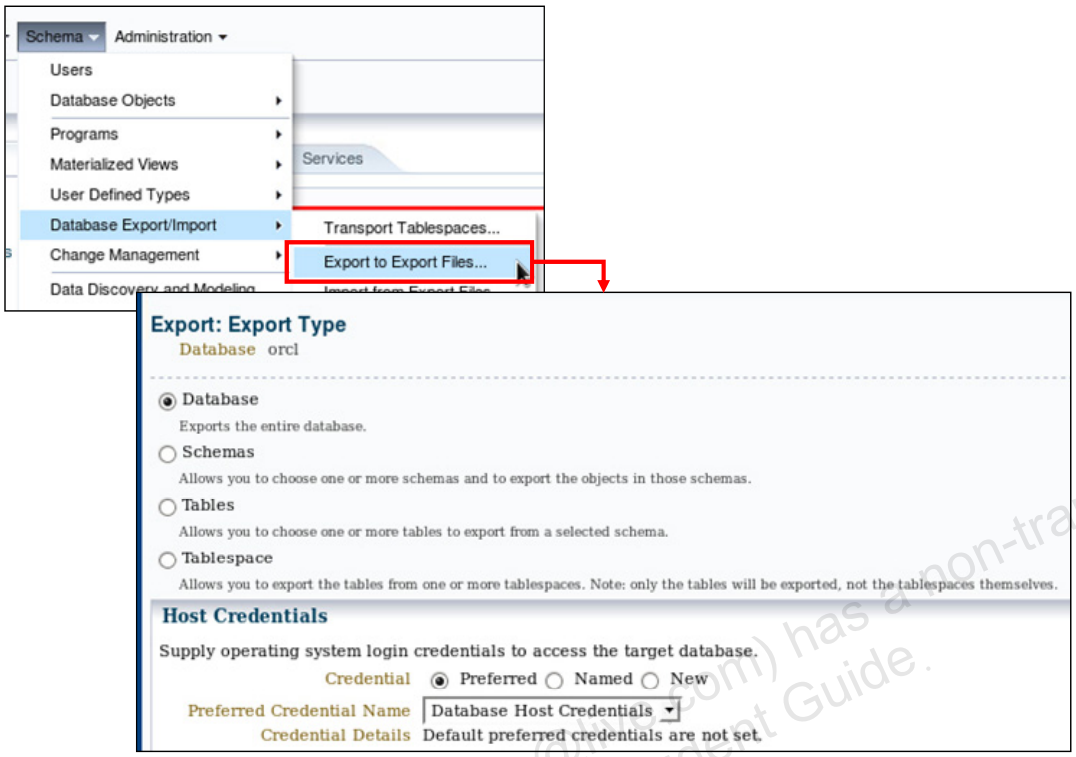
- Data Pump Export and Import interfaces:
  - Command line
  - Parameter file
  - Interactive command line
  - Enterprise Manager Cloud Control
- Data Pump Export and Import modes:
  - Full
  - Schema
  - Table
  - Tablespace
  - Transportable tablespace
  - Transportable database

You can interact with Data Pump Export and Import by using one of the following interfaces:

- **Command-line interface:** Enables you to specify most of the export parameters directly on the command line
- **Parameter file interface:** Enables you to specify all command-line parameters in a parameter file. The only exception is the `PARFILE` parameter.
- **Interactive-command interface:** Stops logging to the terminal and displays the export or import prompts, where you can enter various commands. This mode is enabled by pressing Ctrl + C during an export operation that is started with the command-line interface or the parameter file interface. Interactive-command mode is also enabled when you attach to an executing or stopped job.
- **GUI interface:** Select Schema > Database Export/Import. In the menu select the export or import operation you want to execute.

Data Pump Export and Import provide different modes for unloading and loading different portions of the database. The mode is specified on the command line by using the appropriate parameter. The available modes are listed in the slide and are the same as in the original export and import utilities.

Performing a Data Pump Export by Using Enterprise Manager Cloud Control



Enterprise Manager Cloud Control provides a wizard to guide you through the process of performing a Data Pump export and import procedure. The example in the slide shows a Data Pump export.

From the Database home page, expand the Schema menu, select Database Export/Import, and then select Export to Export Files to begin a Data Pump export session.

The next window that appears is the selection of export type. If a privileged user is connected to the database instance, then the export types include the following:

- Database
- Schemas
- Tables
- Tablespace

If a non-administrative account is used, then the export type list is limited to the following:

- Schemas
- Tables

Click Continue to proceed with the export.

Performing a Data Pump Import

Data Pump can be invoked on the command line:

```
$ impdp hr DIRECTORY=DATA_PUMP_DIR \
DUMPFILE=HR_SCHEMA.DMP \
PARALLEL=1 \
CONTENT=ALL \
TABLES="EMPLOYEES" \
LOGFILE=DATA_PUMP_DIR:import_hr_employees.log \
JOB_NAME=importHR \
TRANSFORM=STORAGE:n
```


Oracle Database also includes Data Pump command-line clients for import and export operations. The example in the slide illustrates a Data Pump import using the `impdp` utility. There are some parameters that are available only by using the command-line interface. For a complete list of options, refer to *Oracle Database Utilities*.

Data Pump Import: Transformations

You can remap:

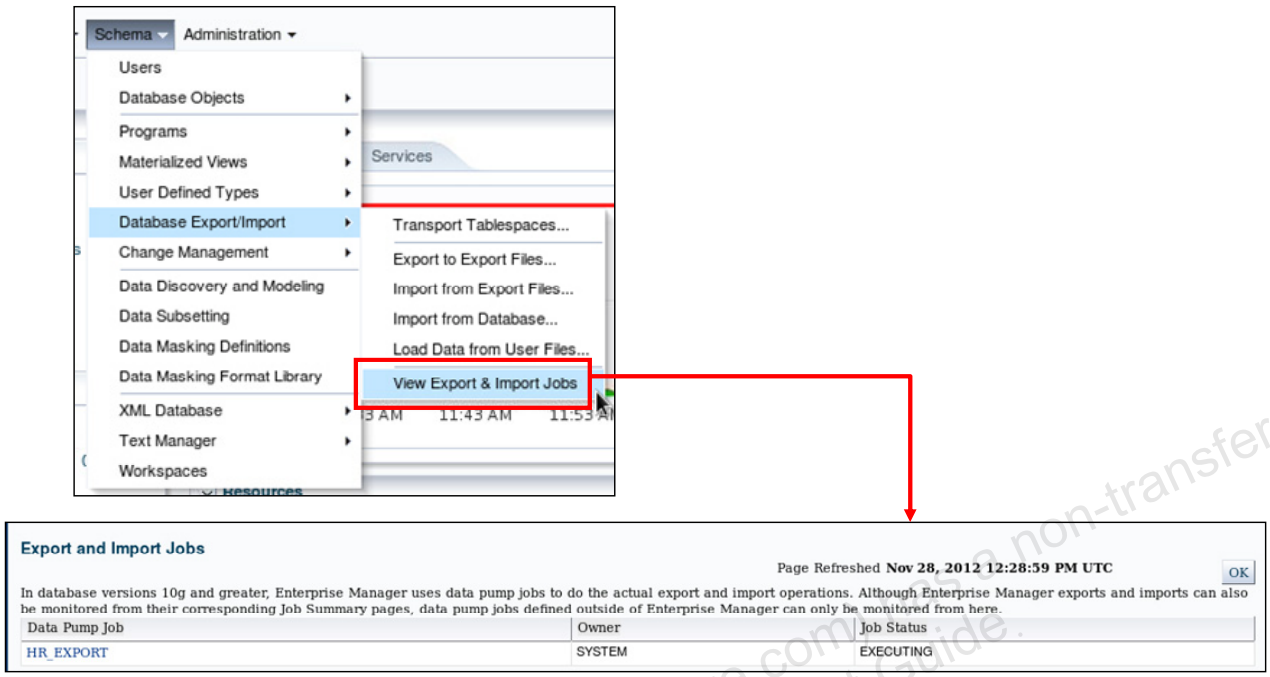
- Data files by using `REMAP_DATAFILE`
- Tablespaces by using `REMAP_TABLESPACE`
- Schemas by using `REMAP_SCHEMA`
- Tables by using `REMAP_TABLE`
- Data by using `REMAP_DATA`

```
REMAP_TABLE = 'EMPLOYEES':'EMP'
```


Because object metadata is stored as XML in the dump file set, it is easy to apply transformations when DDL is being formed during import. Data Pump Import supports several transformations:

- `REMAP_DATAFILE` is useful when moving databases across platforms that have different file-system semantics.
- `REMAP_TABLESPACE` enables objects to be moved from one tablespace to another.
- `REMAP_SCHEMA` provides the old `FROMUSER` /`TOUSER` capability to change object ownership.
- `REMAP_TABLE` provides the ability to rename entire tables.
- `REMAP_DATA` provides the ability to remap data as it is being inserted.

Using Enterprise Manager Cloud Control to Monitor Data Pump Jobs

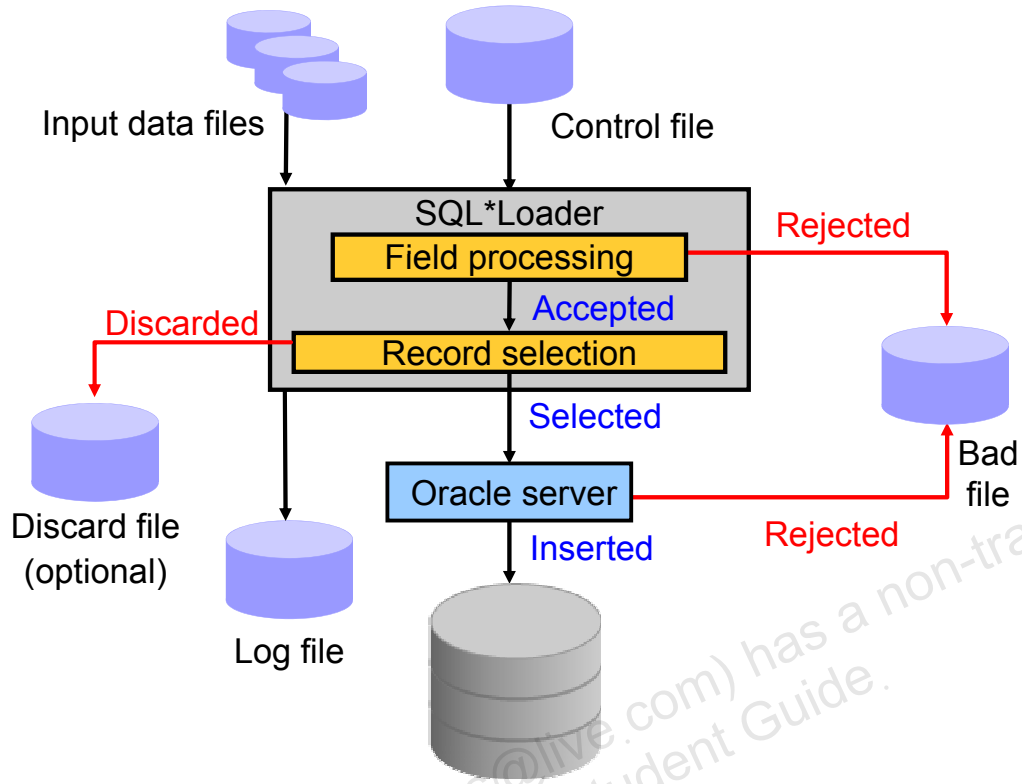


You can use the Enterprise Manager graphical user interface (GUI) to monitor all Data Pump jobs, including those created by using the `expdp` or `impdp` command-line interfaces or by using the `DBMS_DATAPUMP` package.

You can view the current status of the job and change the status to `EXECUTE`, `STOP`, or `SUSPEND`.

To access the “Export and Import Jobs” page, select View Export and Import Jobs in the Database Export/Import menu.

SQL*Loader: Overview



SQL*Loader loads data from external files into tables of an Oracle database. It has a powerful data parsing engine that puts little limitation on the format of the data in the data file.

SQL*Loader uses the following files:

Input data files: SQL*Loader reads data from one or more files (or operating system equivalents of files) that are specified in the control file. From SQL*Loader's perspective, the data in the data file is organized as records. A particular data file can be in fixed record format, variable record format, or stream record format. The record format can be specified in the control file with the `INFILE` parameter. If no record format is specified, the default is stream record format.

Control file: The control file is a text file that is written in a language that SQL*Loader understands. The control file indicates to SQL*Loader where to find the data, how to parse and interpret the data, where to insert the data, and so on. Although not precisely defined, a control file can be said to have three sections.

- The first section contains such session-wide information as the following:
  - Global options, such as the input data file name and records to be skipped
  - `INFILE` clauses to specify where the input data is located
  - Data to be loaded

- The second section consists of one or more `INTO TABLE` blocks. Each of these blocks contains information about the table (such as the table name and the columns of the table) into which the data is to be loaded.
- The third section is optional and, if present, contains input data.

Log file: When SQL*Loader begins execution, it creates a log file. If it cannot create a log file, execution terminates. The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.

Bad file: The bad file contains records that are rejected, either by SQL*Loader or by the Oracle database. Data file records are rejected by SQL*Loader when the input format is invalid. After a data file record is accepted for processing by SQL*Loader, it is sent to the Oracle database for insertion into a table as a row. If the Oracle database determines that the row is valid, the row is inserted into the table. If the row is determined to be invalid, the record is rejected and SQL*Loader puts it in the bad file.

Discard file: This file is created only when it is needed and only if you have specified that a discard file should be enabled. The discard file contains records that are filtered out of the load because they do not match any record-selection criteria specified in the control file.

For more information about SQL*Loader, see the *Oracle Database Utilities* guide.

SQL*Loader Control File

The SQL*Loader control file instructs SQL*Loader about:

- Location of the data to be loaded
- Data format
- Configuration details:
  - Memory management
  - Record rejection
  - Interrupted load handling details
- Data manipulation details

The SQL*Loader control file is a text file that contains data definition language (DDL) instructions. DDL is used to control the following aspects of a SQL*Loader session:

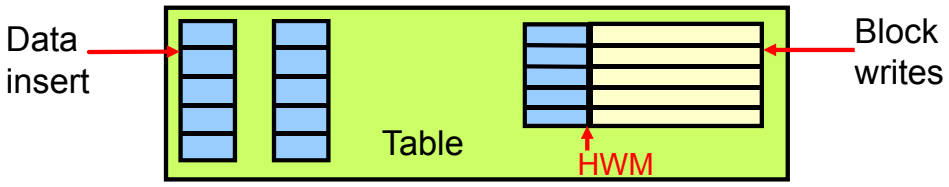
- Where SQL*Loader finds the data to load
- How SQL*Loader expects that data to be formatted
- How SQL*Loader is being configured (including memory management, selection and rejection criteria, interrupted load handling, and so on) as it loads the data
- How SQL*Loader manipulates the data being loaded

```
1   -- This is a sample control file
2  LOAD DATA
3  INFILE 'SAMPLE.DAT'
4  BADFILE 'sample.bad'
5  DISCARDFILE 'sample.dsc'
6  APPEND
7  INTO TABLE emp
8  WHEN (57) = '.'
9  TRAILING NULLCOLS
10 (hiredate SYSDATE,
          deptno POSITION(1:2) INTEGER EXTERNAL(3)
        NULLIF deptno=BLANKS,
          job POSITION(7:14) CHAR TERMINATED BY WHITESPACE
          NULLIF job=BLANKS "UPPER(:job)",
          mgr POSITION(28:31) INTEGER EXTERNAL
          TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
        ename POSITION(34:41) CHAR
          TERMINATED BY WHITESPACE "UPPER(:ename)",
          empno POSITION(45) INTEGER EXTERNAL
          TERMINATED BY WHITESPACE,
          sal POSITION(51) CHAR TERMINATED BY WHITESPACE
          "TO_NUMBER(:sal,'$99,999.99')",
          comm INTEGER EXTERNAL ENCLOSED BY '(' AND '%'
          ":comm * 100"
    )
```

The explanation of this sample control file (by line numbers) is as follows:

1. Comments can appear anywhere in the command section of the file, but they must not appear in the data. Precede any comment with two hyphens. All text to the right of the double hyphen is ignored until the end of the line.
2. The `LOAD DATA` statement indicates to SQL*Loader that this is the beginning of a new data load. If you are continuing a load that has been interrupted in progress, use the `CONTINUE LOAD DATA` statement.
3. The `INFILE` keyword specifies the name of a data file containing data that you want to load.
4. The `BADFILE` keyword specifies the name of a file into which rejected records are placed.
5. The `DISCARDFILE` keyword specifies the name of a file into which discarded records are placed.
6. The `APPEND` keyword is one of the options that you can use when loading data into a table that is not empty. To load data into a table that is empty, use the `INSERT` keyword.
7. The `INTO TABLE` keyword enables you to identify tables, fields, and data types. It defines the relationship between records in the data file and tables in the database.
8. The `WHEN` clause specifies one or more field conditions that each record must match before SQL*Loader loads the data. In this example, SQL*Loader loads the record only if the 57th character is a decimal point. That decimal point delimits dollars and cents in the field and causes records to be rejected if `SAL` has no value.
9. The `TRAILING NULLCOLS` clause prompts SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
10. The remainder of the control file contains the field list, which provides information about column formats in the table that is being loaded.

Loading Methods



Conventional Load	Direct Path Load
Uses COMMIT	Uses data saves (faster operation)
Always generates redo entries	Generates redo only under specific conditions
Enforces all constraints	Enforces only PRIMARY KEY, UNIQUE, and NOT NULL
Fires INSERT triggers	Does not fire INSERT triggers
Can load into clustered tables	Does not load into clusters
Allows other users to modify tables during load operation	Prevents other users from making changes to tables during load operation
Maintains index entries on each insert	Merges new index entries at the end of the load

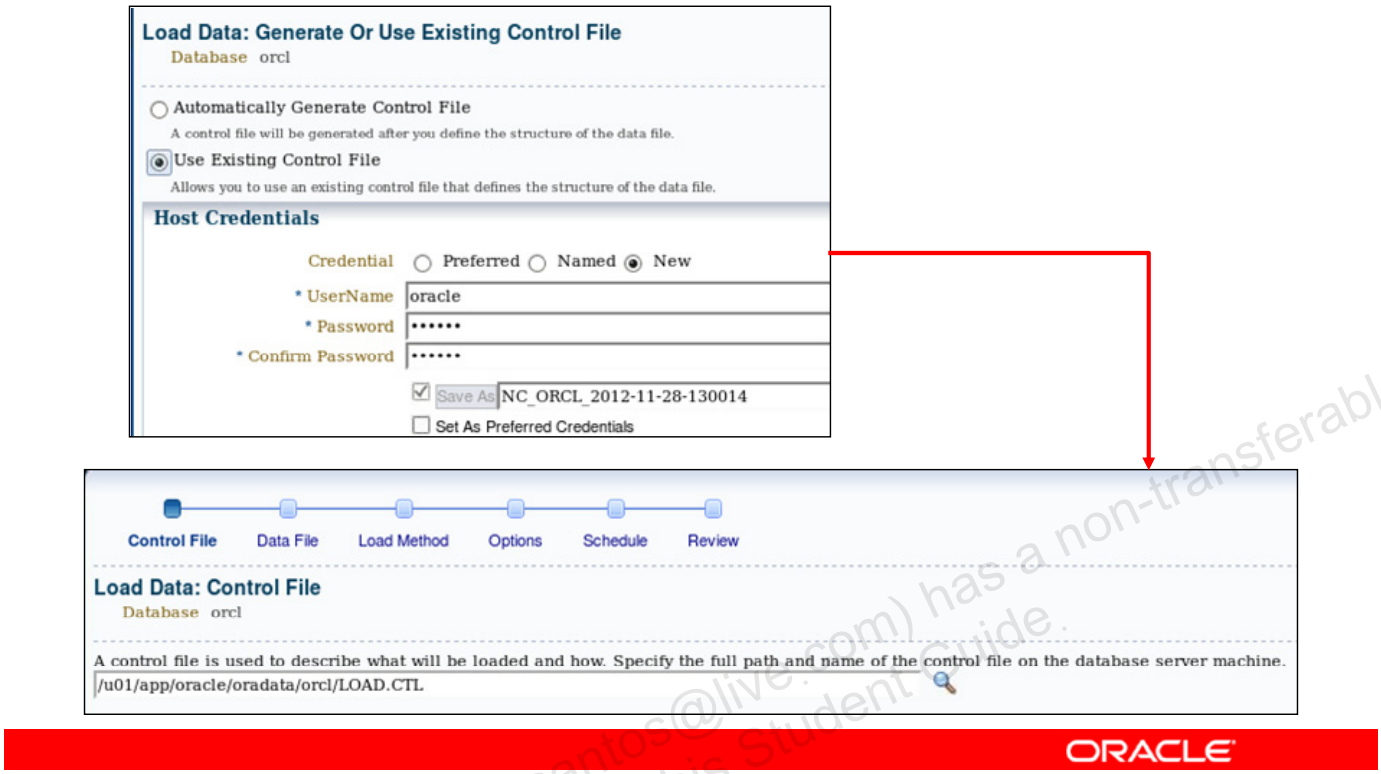
Method of Saving Data

A conventional path load executes SQL INSERT statements to populate tables in an Oracle database. A direct path load eliminates much of the Oracle database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files. A direct load does not compete with other users for database resources, so it can usually load data at close to disk speed. Conventional path loads use SQL processing and a database COMMIT operation for saving data. The insertion of an array of records is followed by a COMMIT operation. Each data load may involve several transactions.

Direct path loads use data saves to write blocks of data to Oracle data files. This is why the direct path loads are faster than the conventional ones. The following features differentiate a data save from COMMIT:

- During a data save, only full database blocks are written to the database.
- The blocks are written after the high-water mark (HWM) of the table.
- After a data save, the HWM is moved.
- Internal resources are not released after a data save.
- A data save does not end the transaction.
- Indexes are not updated at each data save.

Loading Data by Using Enterprise Manager Cloud Control



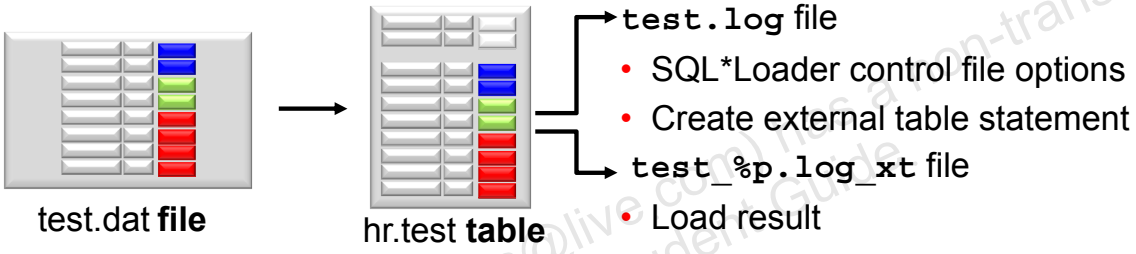
You can load data by using Enterprise Manager Cloud Control. In the Schema menu, select Database Export/Import. Then select Load Data From User Files.

On the first page you indicate whether you are using an existing control file or want a new control file to be generated. Click Continue on this page to invoke the wizard.

SQL*Loader Express Mode

- Specify a table name to initiate an Express Mode load.
- Table columns must be scalar data types (character, number, or datetime).
- A data file can contain 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 several 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).

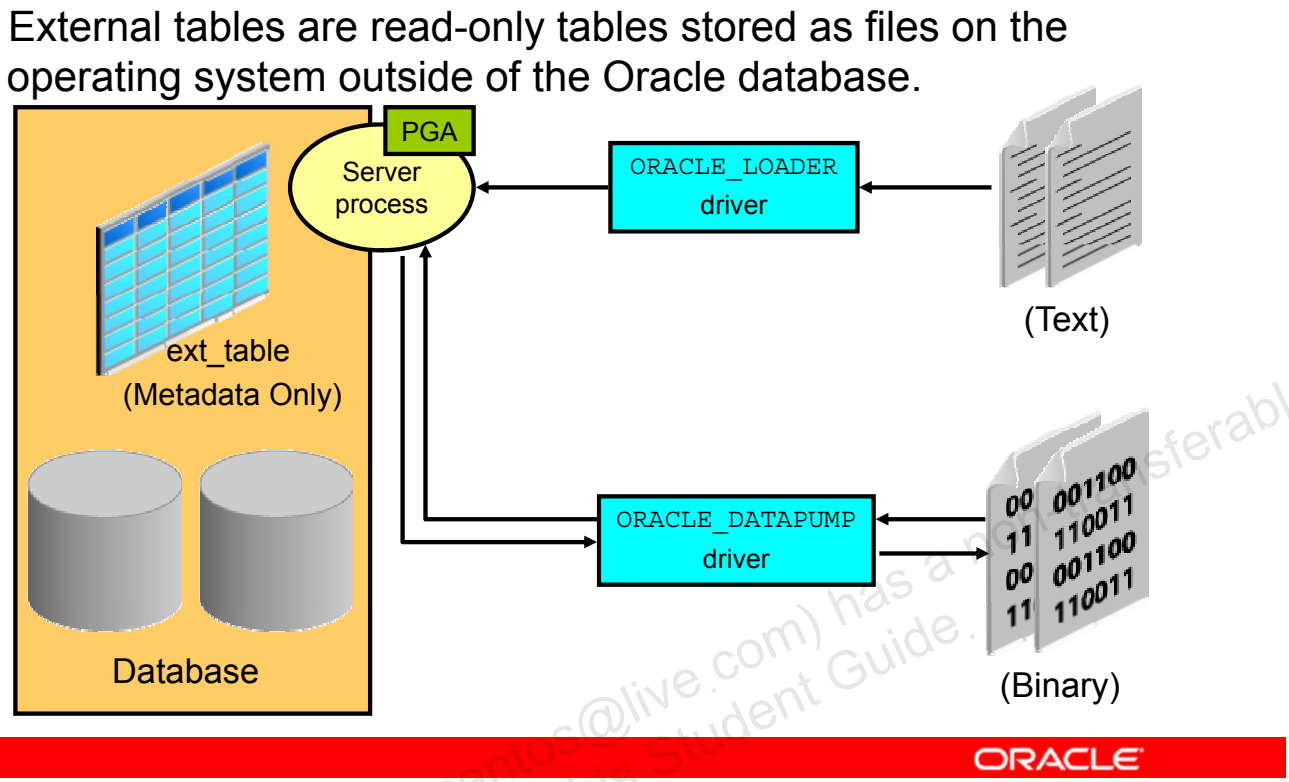
- A log file that includes:
  - The control file output
  - A SQL script for creating the external table and performing the load using a SQL `INSERT AS SELECT` statement

  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.

The following is 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.
```

External Tables



External tables access data in external sources as if it were in a table in the database. You can connect to the database and create metadata for the external table using DDL. The DDL for an external table consist of two parts: one part that describes the Oracle Database column types, and another part that describes the mapping of the external data to the Oracle Database data columns.

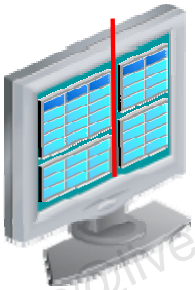
An external table does not describe any data that is stored in the database. Nor does it describe how data is stored in the external source. Instead, it describes how the external table layer must present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the external file so that it matches the external table definition. External tables are read only; therefore, no DML operations are possible, and no index can be created on them.

There are two access drivers used with external tables. The `ORACLE_LOADER` access driver can be used only to read table data from an external table and load it into the database. It uses text files as the data source. The `ORACLE_DATAPUMP` access driver can both load table data from an external file into the database and also unload data from the database into an external file. It uses binary files as the external files. The binary files have the same format as the files used by the Data Pump Import and Export utilities and can be interchanged with them.

External Table: Benefits

- Data can be used directly from the external file or loaded into another database.
- External data can be queried and joined directly in parallel with tables residing in the database, without requiring it to be loaded first.
- The results of a complex query can be unloaded to an external file.
- You can combine generated files from different sources for loading purposes.

From Oracle Database



From external file

The data files created for the external table can be moved and used as the data files for another external table in the same database or different database. External data can be queried and joined directly in parallel to tables residing in the database, without requiring the data to be loaded first. You can choose to have your applications directly access external tables with the `SELECT` command, or you can choose to have data loaded first into a target database.

The results of a complex query can be unloaded to an external file using the `ORACLE_DATAPUMP` access driver.

Data files that are populated by different external tables can all be specified in the `LOCATION` clause of another external table. This provides an easy way of aggregating data from multiple sources. The only restriction is that the metadata for all the external tables must be exactly the same.

Defining an External Tables with ORACLE_LOADER

```
CREATE TABLE extab_employees
                (employee_id       NUMBER(4),
                 first_name        VARCHAR2(20),
                 last_name         VARCHAR2(25),
                 hire_date         DATE)
ORGANIZATION EXTERNAL
    ( TYPE ORACLE_LOADER DEFAULT DIRECTORY extab_dat_dir
      ACCESS PARAMETERS
      ( records delimited by newline
        badfile extab_bad_dir:'empxt%a_%p.bad'
        logfile extab_log_dir:'empxt%a_%p.log'
        fields terminated by ','
        missing field values are null
    ( employee_id, first_name, last_name,
     hire_date char date_format date mask "dd-mon-yyyy"))
    LOCATION ('empxt1.dat', 'empxt2.dat') )
    PARALLEL  REJECT LIMIT UNLIMITED;
```


The metadata for an external table is created using the SQL language in the database. The ORACLE_LOADER access driver uses the SQL*Loader syntax to define the external table. This command does not create the external text files.

The example in the slide shows three directory objects (EXTAB_DAT_DIR, EXTAB_BAD_DIR, and EXTAB_LOG_DIR) that are created and mapped to existing OS directories to which the user is granted access.

When the EXTAB_EMPLOYEES table is accessed, SQL*Loader functionality is used to load the table, and at that instance the log file and bad file are created.

Best-practice tip: If you have a lot of data to load, enable PARALLEL for the load operation:

```
ALTER SESSION ENABLE PARALLEL DML;
```

External Table Population with ORACLE_DATAPUMP

```
CREATE TABLE ext_emp_query_results
  (first_name, last_name, department_name)
ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY ext_dir
    LOCATION ('emp1.exp','emp2.exp','emp3.exp')
  )
PARALLEL
AS
SELECT e.first_name,e.last_name,d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id AND
       d.department_name in
                    ('Marketing', 'Purchasing');
```


This example shows you how the external table population operation can help to export a selective set of records resulting from the join of the EMPLOYEES and DEPARTMENTS tables.

Because the external table can be large, you can use a parallel populate operation to unload your data to an external table. As opposed to a parallel query from an external table, the degree of parallelism of a parallel populate operation is constrained by the number of concurrent files that can be written to by the access driver. There is never more than one parallel execution server writing into one file at a particular point in time.

The number of files in the LOCATION clause must match the specified degree of parallelism because each input/output (I/O) server process requires its own file. Any extra files that are specified are ignored. If there are not enough files for the specified degree of parallelism, the degree of parallelization is lowered to match the number of files in the LOCATION clause.

The external table is read-only after is has been populated. The SELECT command can be very complex, allowing specific information to be populated in the external table. The external table, having the same file structure as binary Data Pump files, can then be migrated to another system, and imported with the impdp utility or read as an external table.

Note: For more information about the ORACLE_DATAPUMP access driver parameters, see the *Oracle Database Utilities* guide.

Using External Tables

- Querying an external table:

```
SQL> SELECT * FROM extab_employees;
```

- Querying and joining an external table with an internal table:

```
SQL> SELECT e.employee_id, e.first_name, e.last_name,
d.department_name FROM departments d, extab_employees e
WHERE d.department_id = e.department_id;
```

- Appending data to an internal table from an external table:

```
SQL> INSERT /*+ APPEND */ INTO hr.employees SELECT *
FROM extab_employees;
```


External tables are queried just like internal database tables. The first example illustrates querying an external table named `EXTAB_EMPLOYEES` and only displaying the results. The results are not stored in the database.

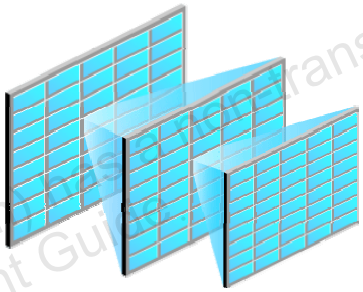
The second example shows the joining of an internal table named `DEPARTMENTS` with an external table named `EXTAB_EMPLOYEES` and only displaying the results.

The third example in the slide illustrates the direct appending of an internal table data with the query and load of data from an external table.

Data Dictionary

View information about external tables in:

- [DBA| ALL| USER]_EXTERNAL_TABLES
- [DBA| ALL| USER]_EXTERNAL_LOCATIONS
- [DBA| ALL| USER]_TABLES
- [DBA| ALL| USER]_TAB_COLUMNS
- [DBA| ALL]_DIRECTORIES



The data dictionary views in the slide provide information about external tables:

[DBA| ALL| USER]_EXTERNAL_TABLES: Specific attributes of external tables in the database

[DBA| ALL| USER]_EXTERNAL_LOCATIONS: Data sources for external tables

[DBA| ALL| USER]_TABLES: Descriptions of the relational tables in the database

[DBA| ALL| USER]_TAB_COLUMNS: Descriptions of the columns of tables, views, and clusters in the database

[DBA| ALL]_DIRECTORIES: Descriptions of the directory objects in the database

Quiz

Like other database objects, directory objects are owned by the user that creates them unless another schema is specified during creation.

a. True
b. False

Answer: b

Quiz

An index can be created on an external table.

a. True
b. False

Answer: b

Summary

In this lesson, you should have learned how to:

- Describe ways to move data
- Explain the general architecture of Oracle Data Pump
- Create and use directory objects
- Use Data Pump Export and Import to move data between Oracle databases
- Use SQL*Loader to load data from a non-Oracle database (or user files)
- Use external tables to move data via platform-independent files

Practice 19

- 19-1: Moving Data by Using Data Pump
- 19-2: Loading Data by Using SQL*Loader