# Administration des bases de données
# Chapitre 6 : Déplacement des données

**Ines BAKLOUTI**

ines.baklouti@esprit.tn

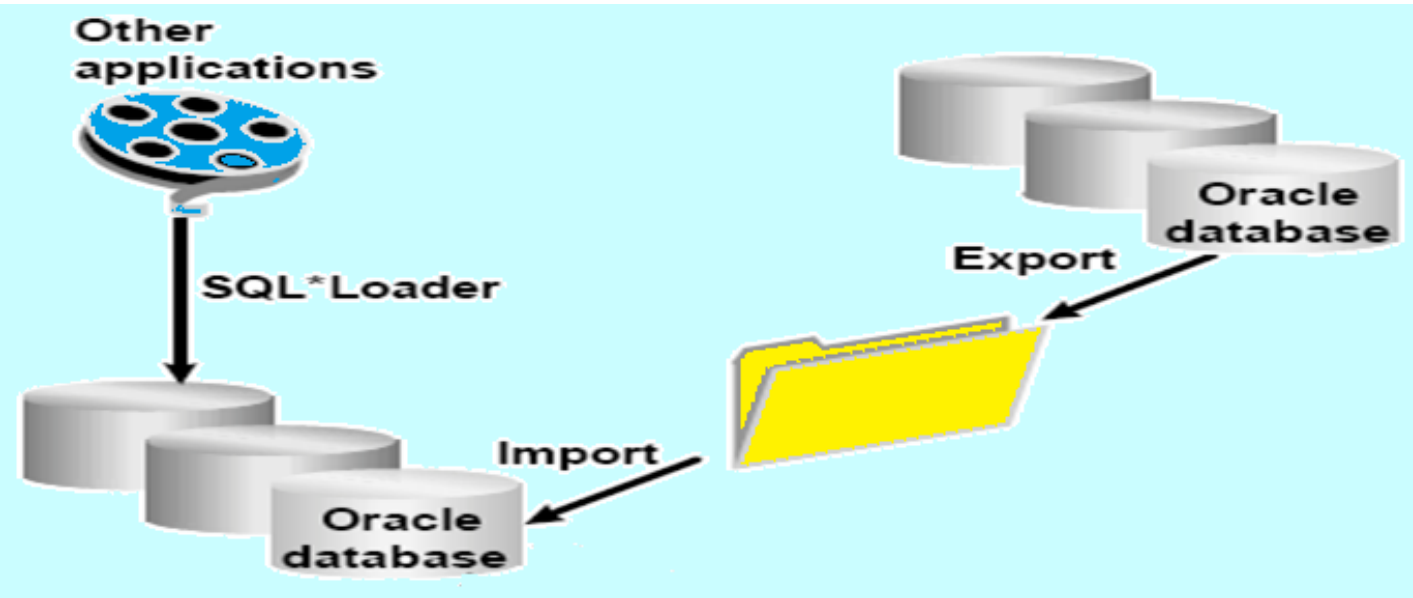
Ecole Supérieure Privée d'Ingénierie et de Technologies

# Plan

1. Oracle Data Pump
   - EXPDP / IMPDP
   - Package DBMS_DATAPUMP
2. SQL*Loader
   - Log File
   - Bad File
   - Discard File
   - Control File

# Introduction

- Il existe plusieurs méthodes pour charger des données dans les tables d'une base de données Oracle :
  - SQL*Loader
  - Utilitaire d'export / import ORACLE DATA PUMP



# Plan

# Oracle Data Pump

- Les méthodes d'export/import classiques se sont révélées très longues : pour 33 Gb de données : 3h30 d'export et 7 à 8h d'import !

- Oracle a introduit avec la version 10g un nouvel utilitaire d'export/import appelée **EXPORT / IMPORT DATA PUMP**
- Oracle Data Pump est un utilitaire (coté serveur) de déplacement de données et de méta-données de masse de manière très rapide entre des bases de données Oracle
- Oracle Data Pump fournit deux nouveaux utilitaires d'export et import (expdp et impdp) qui offrent une meilleur performance en utilisant le traitement parallèle

# Lancement du Data Pump

**Notez Bien**

Avant de pouvoir utiliser Data Pump, vous devez créer un Directory Oracle où vous allez stocker vos exports.

- Data Pump peut être appelé :
  - via Enterprise Manager (EM) Database Control
  - par les binaires EXPDP et IMPDP situés dans le dossier bin de ORACLE_HOME
  - via le package SYS.DBMS_DATAPUMP

# Modes d'export / import avec Data Pump

Il existe 4 modes d'export/import avec Data Pump :

- Export / Import COMPLET
  - Operation demandée par le paramètre FULL
- Export / Import de SCHEMA (metadata)
  - Mode par defaut
  - Permet l'export ou l'import d'un ou plusieurs schemas de la base de données
- Export / Import de TABLE
  - Mode commandé par le parametre TABLES
  - Permet de sélectionner des tables à exporter à partir d'un schema
- Export / Import de TABLESPACE
  - Permet d'exporter les tables d'au moins un espace disque logique
    Remarque : seules les tables seront exportées, et non les espaces disque logiques eux-memes

# Fichiers générés avec Data Pump

- 3 types de fichiers sont générés avec Data Pump
  - Fichiers SQL : contiennent les instructions LDD de la création des objets de la base de données
  - Fichiers DUMP : contiennent les données exportées
  - Fichiers LOG : contiennent le journal d'historique de l'exécution du job (export ou import)

# Privilèges nécessaires

- Deux privilèges sont nécessaires pour l'export / import :
  - Le privilège EXP_FULL_DATABASE et IMP_FULL_DATABASE
    Ces privilèges permettent d'exporter l'intégralité d'une BD, un Tablespace, ou encore un schéma autre que le sien ou bien une table située dans un autre schéma
  - Tous les utilisateurs qui possèdent le rôle DBA ont la possibilité d'effectuer un import / export d'une base de données

# Paramètres d'export du Data Pump : expdp help=y

| Mot clé | Description |
|---|---|
| CONTENT | Specifies data to unload where the valid keywords are: (ALL), DATA_ONLY, and METADATA_ONLY |
| DIRECTORY | Directory object to be used for dumpfiles and logfiles |
| DUMPFILE | List of destination dump files (expdat.dmp), e.g. DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp. |
| EXCLUDE | Exclude specific object types, e.g. EXCLUDE=TABLE:EMP |
| FILESIZE | Specify the size of each dumpfile in units of bytes |
| FULL | Export entire database (N) |
| INCLUDE | Include specific object types, e.g. INCLUDE=TABLE_DATA |
| SCHEMAS | List of schemas to export (login schema) |
| TABLES | Identifies a list of tables to export - one schema only |
| TABLESPACES | Identifies a list of tablespaces to export |
| TRANSPORT_FULL_CHECK | Verify storage segments of all tables (N) |
| TRANSPORT_TABLESPACES | List of tablespaces from which metadata will be unloaded |
| COMPRESSION | Réduction de la taille du contenu du fichier de vidage, les mots-clés valides étant : (METADATA_ONLY) et NONE |
| ENCRYPTION_PASSWORD | Clé de mot de passe pour la création de données de colonne cryptées |

# Paramètres d'import du Data Pump : impdp help=y

| Mot clé | Description |
|---|---|
| CONTENT | Specifies data to unload where the valid keywords are: (ALL), DATA_ONLY, and METADATA_ONLY |
| DIRECTORY | Directory object to be used for dumpfiles and logfiles |
| DUMPFILE | List of destination dump files (expdat.dmp), e.g. DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp |
| EXCLUDE | Exclude specific object types, e.g. EXCLUDE=TABLE:EMP |
| FILESIZE | Specify the size of each dumpfile in units of bytes |
| FULL | import entire database (N) |
| INCLUDE | Include specific object types, e.g. INCLUDE=TABLE_DATA |
| SCHEMAS | List of schemas to export (login schema) |
| TABLES | Identifies a list of tables to import |
| TABLESPACES | Identifies a list of tablespaces to import |
| TRANSPORT_FULL_CHECK | Verify storage segments of all tables (N) |
| TRANSPORT_TABLESPACES | Import des metadonnees de tablespaces transportables |
| REMAP_TABLESPACES | Association de l'objet à un autre tablespace |
| PARALLEL | Modification du nombre de processus de travail actifs pour le job en cours |

# EXPDP / IMPDP

## Exemple

1. **Création d'un utilisateur exp_imp et accord de privilèges nécessaires :**
   SQL> create user exp_imp identified by expimp ;
   SQL> grant connect, resource to exp_imp ;
   SQL> grant exp_full_database, imp_full_database to exp_imp ;

2. **Création d'un répertoire pour stocker les fichiers d'export et les fichiers log : DIRECTORY oracle**
   SQL> connect sys/sys as sysdba
   SQL> create or replace directory DATAPUMP as 'C :\oraclexe' ;
   SQL> grant read, write on directory DATAPUMP to exp_imp ;

3. **Création de la table à exporter**
   SQL> create table test( a number ) ;
   SQL> insert into test values (1) ;
   SQL> insert into test values (2) ;
   SQL> insert into test values (3) ;
   SQL> commit ;

4. **Export de la table test du schema exp_imp**
   SQL> host
   C :\> expdp exp_imp/expimp tables=exp_imp.test DIRECTORY=DATAPUMP DUMPFILE=test.dump LOGFILE=test.log
   => Création de 2 fichiers test.dump + test.log sous Oracle Directory 'C :\oraclexe'

5. **Importer les données dans la table test**
   SQL> host
   C :\> impdp exp_imp/expimp content=DATA_ONLY tables=exp_imp.test DIRECTORY=DATAPUMP DUMPFILE=test.dump LOGFILE=test.log
   => Ajout des 3 lignes déjà existantes encore une fois dans la table test

6. **Vérification que l'import a réussi**
   C :\> exit ( quitter cmd et revenir à SQL)
   SQL> select * from test ;

# Vue d'export / import

- Pour vérifier le déroulement du job d'export ou d'import, on peut interroger la vue DBA_DATAPUMP_JOBS

```
SQL> SELECT * FROM DBA_DATAPUMP_JOBS ;

OWNER_NAME   JOB_NAME              OPERATION       JOB_MODE     STATE
------------------------- ------------------------------ ---------
DVP         SYS_IMPORT_TABLE_01   IMPORT         TABLE      DEFINING
```

# Package DBMS_DATAPUMP

## Procédures

| Procedure Name | Valid States | Description |
|---|---|---|
| ADD_FILE | Defining (valid for both export and import jobs)<br><br>Executing and Idling (valid only for specifying dump files for export jobs) | Specifies a file for the dump file set, the log file, or the SQL_FILE output. |
| ATTACH | Defining, Executing, Idling, Stopped, Completed, Completing, Not Running | Allows a user session to monitor a job or to restart a stopped job. The attach will fail if the dump file set or master table for the job have been deleted or altered in any way. |
| DATA_FILTER | Defining | Restricts data processed by a job. |
| DETACH | All | Disconnects a user session from a job. |
| GET_DUMPFILE_INFO | All | Retrieves dump file header information |
| GET_STATUS | All, except Completed, Not Running, Stopped, and Undefined | Obtains the status of a job. |
| LOG_ENTRY | Defining, Executing, Idling, Stop Pending, Completing | Adds an entry to the log file. |
| METADATA_FILTER | Defining | Restricts metadata processed by a job. |
| METADATA_REMAP | Defining | Remaps metadata processed by a job. |
| METADATA_TRANSFORM | Defining | Alters metadata processed by a job. |
| OPEN | Undefined | Creates a new job. |
| SET_PARALLEL | Defining, Executing, Idling | Specifies parallelism for a job. |
| SET_PARAMETER | Defining | Alters default processing by a job. |
| START_JOB | Defining, Idling | Begins or resumes execution of a job. |
| STOP_JOB | Defining, Executing, Idling, Stop Pending | Initiates shutdown of a job. |
| WAIT_FOR_JOB | All, except Completed, Not Running, Stopped, and Undefined | Waits for a job to end. |

# Package DBMS_DATAPUMP

## Exemple

### 1. Programmer l'export :

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
p_dph NUMBER;
BEGIN
p_dph := DBMS_DATAPUMP.open(operation=>'EXPORT', job_mode=>'TABLE',
job_name=>'DVP_EXPORT');
DBMS_DATAPUMP.add_file(handle=>p_dph, filename=>'dvp.dmp', directory=> 'DATAPUMP',
filetype=>1);
DBMS_DATAPUMP.add_file(handle=>p_dph, filename=>'tab_dvp.log', directory=>
'DATAPUMP', filetype=>3);
DBMS_DATAPUMP.metadata_filter(handle =>p_dph, name=>'SCHEMA_EXPR', value =>'IN
("DVP")');
DBMS_DATAPUMP.start_job(p_dph);
DBMS_DATAPUMP.detach(p_dph);
dbms_output.put_line('Export terminé ');
EXCEPTION
WHEN OTHERS THEN
dbms_output.put_line('erreur :'||sqlerrm||' Job-ID :'||p_dph);
END;
/

Export terminé
Procédure PL/SQL terminée avec succès
```

# Package DBMS_DATAPUMP

## Exemple

### 2. Programmer l'import :

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE p_dph NUMBER;
BEGIN
p_dph :=DBMS_DATAPUMP.open(operation=>'IMPORT', job_mode=>'TABLE',
job_name=>'EMP_IMPORT2');
DBMS_DATAPUMP.add_file(handle=>p_dph, filename=>'dvp.dmp', directory =>'DATAPUMP',
filetype=>1);
DBMS_DATAPUMP.add_file(handle=>p_dph, filename=>'dvp_imp.log', directory =>
'DATAPUMP', filetype=>3);
DBMS_DATAPUMP.set_parameter(handle=> p_dph, name=>'TABLE_EXISTS_ACTION',
value=>'REPLACE');
DBMS_DATAPUMP.start_job(p_dph);
DBMS_DATAPUMP.detach(p_dph);
dbms_output.put_line('Import terminé ');
EXCEPTION WHEN OTHERS THEN
dbms_output.put_line('erreur :'||sqlerrm||' Job-ID :'||p_dph);
END;
/
```

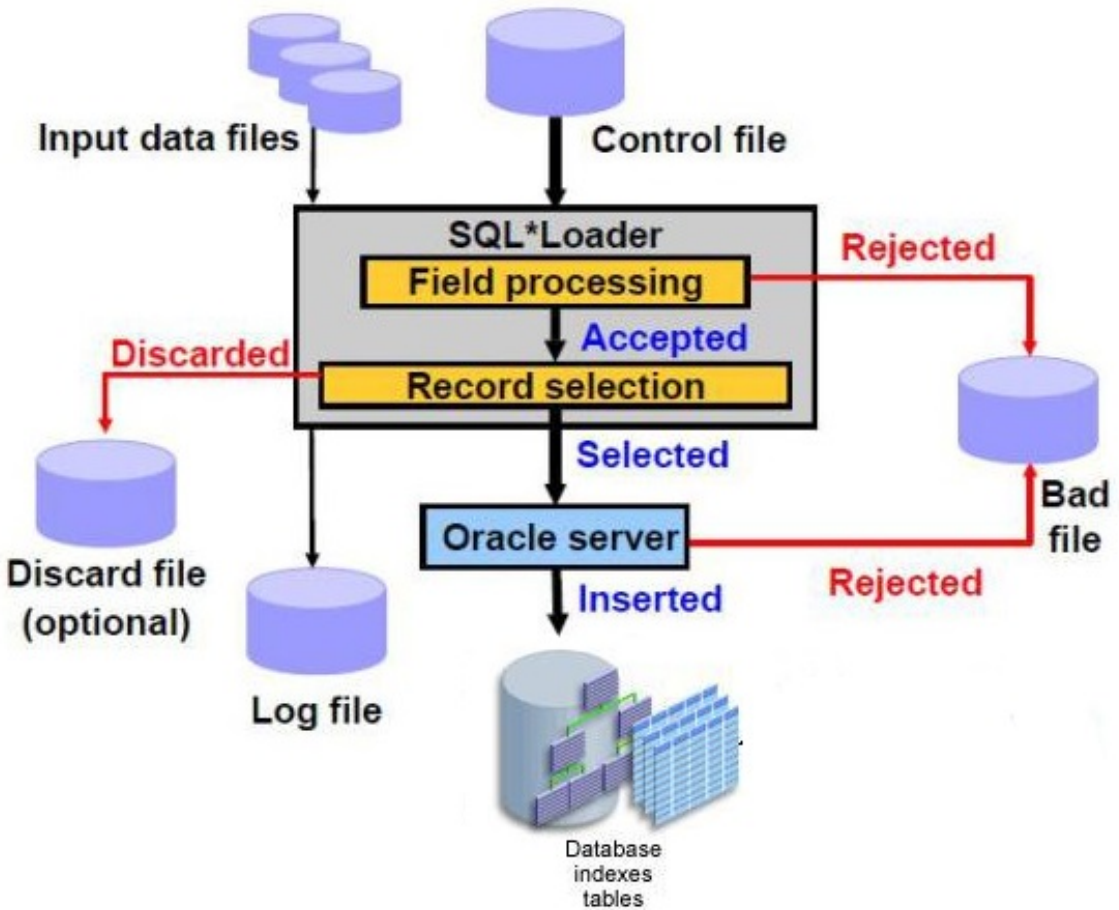
Import terminé

Procédure PL/SQL terminée avec succès.

# Plan

# SQL*Loader

- L'utilitaire SQL*Loader charge les données de fichiers externes dans des tables d'une base de données Oracle
- Cet utilitaire dispose d'un puissant moteur d'analyse (parse) des données, qui ne limite que très peu le format des données du fichier



# Log File

- Le fichier LOG FILE enregistre les activités SQL Loader durant un chargement de données :
  - Les noms des fichiers CONTROL FILE, BAD FILE, DISCARD FILE, Input Data File
  - Les champs et types de données qui ont été chargées
  - Messages d'erreurs sur les enregistrement non chargés
  - Le nombre d'enregistrements lus dans le fichier de données
  - Le nombre d'enregistrements rejetés en raison d'erreurs
  - Le nombre d'enregistrements rejetés en raison de critères de sélection
  - Le temps de chargement

# Bad File

Dès que SQL Loader rencontre une erreur en essayant de lire ou de charger un enregistrement, il écrit dans un fichier BAD FILE et enregistre les erreurs SQL Loader : enregistrement non conforme au format dans le fichier de contrôle, violations de contraintes d'intégrité, tablespace plein, etc.

# Discard File

Les enregistrements qui ne répondent pas aux conditions de sélection spécifiées dans le Control File sont rejetés, écartés et sont écrits dans le fichier DISCARD FILE.

# Control File

Le fichier de contrôle indique à SQL*Loader :

- L'emplacement des fichiers bad file, discard file et log file
- Les détails de configuration : Gestion de la mémoire Rejet des enregistrements
- L'emplacement des données à charger : input data files
- la structure, types, longueurs, précisions des données à charger
- Les noms de tables à charger
- La correspondance entre les champs des données et les colonnes des tables de base de données
- Les critères de sélection qui définissent les enregistrements à insérer dans les tables de base de données
- Formattage des enregistrements de données : si le format est délimité, retour chariot, si une colonne est de taille fixe, etc.

# Control File

## Exemple 1

- Soit le fichier « Emp.dat » contenant :
  10001,"Scott Tiger", 1000, 40
  10002,"Frank Naude", 500, 20
- Pour charger le fichier emp.dat dans la table « emp » du schéma HR, le control file doit contenir les informations suivantes :
  **load data**
  **infile** 'c :\Emp.dat'
  **into table** emp – { INSERT | REPLACE | TRUNCATE | APPEND }
  **fields terminated by** "," **optionally enclosed by** '"'
  ( empno, empname, sal, deptno )

Pour charger les données on doit exécuter la commande suivante :
SQL> host
C :\> Sqlldr user/password control=<control_file>.ctl
log=<log_file>.log bad=<bad_file>.bad discard=<discard_file>.dsc

# Control File

## Exemple 2 : charger des données à taille fixe

- Exemple 1 :
  - Soit le fichier « dept.txt » contenant :
    COSC COMPUTER SCIENCE
    ENGL ENGLISH LITERATURE
    MATH MATHEMATICS
    POLY POLITICAL SCIENCE
  - Control File
    **load data**
    **infile** 'c :\dept.txt'
    **append into table departments**
    – insérer en fin de table
    ( dept position (01 :04) char(4),
    deptname position (05 :25)
    char(20) )

- Exemple 2 :
  - Control file
    **load data**
    **infile *** – les datas ne proviennent pas d'un file
    **append into table departments**
    – insérer en fin de table
    ( dept position (01 :04) char(4),
    deptname position (05 :25)
    char(20) )
    **begindata**
    COSC COMPUTER SCIENCE
    ENGL ENGLISH LITERATURE
    MATH MATHEMATICS
    POLY POLITICAL SCIENCE

# Control File

## Exemple 3 : convertir les données au moment du chargement

- Soit le fichier « emp.dat » contenant :
  11111AAAAAAAAAA991201
  22222BBBBBBBBB990112
- On veut charger le fichier emp.dat dans une table modified_data contenant les champs suivants :
  - rec_no : un numéro séquentiel
  - region : 31
  - time_loaded : temps de chargement, date du jour
  - data1 : le premier champ numérique dans emp.dat, avec 2 chiffres après la virgule
  - data2 : le deuxième champs en majuscule dans emp.dat
  - data3 : le troisième champ numérique dans emp.dat, date suivant le format « YYMMDD »
- Mettre les mauvais enregistrements dans le fichier « bad.bad »

**Solution :**

- Créer une séquence « db_seq » :
  CREATE SEQUENCE db_seq START WITH 1 INCREMENT BY 1 ;
- Editer un control file :

```
load data
infile 'Emp.dat'
badfile 'bad.bad'
INTO TABLE modified_data
(rec_no " db_seq.nextval ",
region CONSTANT '31',
time_loaded " to_char(SYSDATE, 'HH24 :MI') ",
data1 position(1 :5)  " :data1/100",
data2 position(6 :15)  "upper( :data2)",
data3 position(16 :22)  "to_date( :data3,
'YYMMDD')
)
```

# Control File

## Exemple 4 : chargement sélectif des données

- Soit le fichier « emp.dat » contenant les enregistrements suivants :
  1111AAAAAAAAAA991202
  2222BBBBBBBBBB990112
  3333ABBBBBBBBB990112
  4444ABBBBBBBBB990112
- On veut charger dans la table emp les employés dont :
  - Le nom commence par B
  - La date d'embauche = 1999-01-12
- Solution : le fichier de contrôle aura la forme suivante :
  load data
  infile 'emp.dat'
  append into table emp
  when (05) = 'B' and (15 :20) = '990112' – conditions à vérifier lors du chargement
  trailing nullcols – considérer les colonnes non existantes dans un enregistrement comme NULL
  (empID POSITION (01 :04),
  nom POSITION(05 :14),
  date_emb POSITION(15 :20) )

# Control File

## Exemple 5 : chargement dans plusieurs tables

```
LOAD DATA
INFILE 'C :\emp.dat'

REPLACE INTO TABLE emp
WHEN empno != ' '
( empno POSITION(1 :4) INTEGER,
ename POSITION(6 :15) CHAR,
deptno POSITION(17 :18) CHAR,
mgr POSITION(20 :23) INTEGER )

INTO TABLE proj
WHEN projno != ' '
( projno POSITION(25 :27) INTEGER,
empno POSITION(1 :4) INTEGER )
```