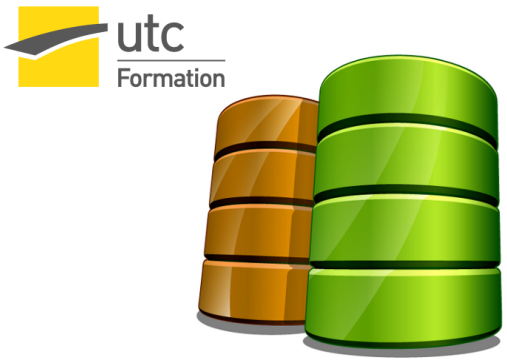
Introduction à l'ETL et application avec Oracle

Data warehouse
http://dwh.crzt.fr



Stéphane Crozat

1 septembre 2016

Table des matières

Objectifs

- Savoir implémenter un data warehouse avec un SGBD relationnel
- Savoir implémenter un processus ETL vers un data warehouse

Introduction

- Volume de cours : 4h
- Volume d'exercice : 12h (hors compléments)

Principes généraux d'un processus ETL

I

1. Principe de l'ETL

Définition : Processus "Extraction, Transformation, Load"

L'ETL * est le processus qui permet de charger un data warehouse à partir de données externes généralement issues de bases transactionnelles. Son rôle est de récupérer ces données et de les traiter pour qu'elles correspondent aux besoins du modèle dimensionnel.

En général les données sources doivent être "nettoyées" et aménagées pour être exploitables par les outils décisionnels.

Fondamental

> You get the data out of its original source location (E), you do something to it (T), and then you load it (L) into a final set of tables for the users to query.
>
> (Kimball et al., 2008, p369) *

Fondamental

Selon Kimball (2004, p.xxi) * 70% de l'effort consacré à un projet de BI est dépensé dans l'ETL.

2. ETL ex nihilo ou outil d'ETL

Un ETL peut être :

- développé *ex nihilo* pour un projet directement dans un langage bas niveau (Java, SQL, PL/SQL...) ;
- ou s'appuyer sur un outil d'ETL (Talend Open Studio, Pentaho Data Integration, Informatica PowerCenter, ...).

Fondamental

« ETL Tool versus Hand Coding (Buy a Tool Suite or Roll Your Own?)

The answer is, "It depends."

(Kimball, Caserta, 2004, pp10-13) * »

Méthode : ETL basés sur un outil

Les avantages offerts par l'outil ETL sont :

- De structurer et de rassembler l'ensemble des morceaux de code nécessaire aux transferts et aux transformations des données
- D'offrir une représentation graphique des flux et opérations
- De faciliter la maintenance et l'évolution de l'ETL
- D'intégrer la gestion des métadonnées
- D'intégrer la gestion des erreurs
- De disposer d'API dédiées (connexion, import/export...) d'accès aux données (CSV *, BD *, XML *...)
- ...

Méthode : ETL ex nihilo

Les avantages offerts par une approche manuelle sont :

- L'homogénéité technologique et la disponibilité interne des compétences : les équipes utilisent les langages qu'elles maîtrisent sans apprentissage et médiation d'un outil tiers.
- La flexibilité : tout est possible.
- Le traitement des fichiers plats (hors BD *) peut être plus simples et plus performant avec des langages proches des systèmes.
- ...

3. ETL en mode batch ou en mode flux

Fondamental : ETL en mode batch

« The standard architecture for an ETL system is based on periodic batch extracts from the source data, which then flows through the system, resulting in a batch update of the final end user tables.

(Kimball, Caserta, 2004, p13) * »

Un ETL alimente en général un data warehouse par des processus *batch* périodiques.

Remarque : ETL en mode flux

Il existe néanmoins des applications nécessitant des data warehouses alimentés en temps réel en mode flux (qui ne sont pas abordés dans le cadre de ce cours).

4. ETL incrémental

Définition : ETL non incrémental

Un ETL non incrémental est :

- soit un ETL qui ne sert qu'une seule fois (*one shot*) ;
- soit un ETL qui refait 100% du processus de migration à chaque fois que l'on souhaite une mise à jour (le data warehouse est vidé puis rempli à nouveau avec les données actuelles)
  On notera qu'un tel ETL *ne gère pas d'historisation.*

Définition : ETL incrémental

Un ETL instrumente normalement un processus incrémental.

1. Les données sont modifiées dans les systèmes transactionnels :
   - mise à jour des dimensions ;
   - ou ajouts de nouveaux faits.
2. L'ETL répercute les mises à jour dans le data warehouse.

Attention : Accumulation des faits

Classiquement les faits s'accumulent dans le data warehouse, il n'y a jamais ni suppression ni mise à jour (croissance monotone).

Attention : Historisation des dimensions

Lorsqu'une dimension est mise à jour, l'ETL doit garder la mémoire des anciennes valeurs afin que les anciens faits restent bien reliés aux anciennes valeurs.

Ils existent plusieurs stratégies pour gérer l'historique des valeurs des dimensions dans le DW :

- associer des dates aux dimensions et aux faits afin de savoir quelle valeur de dimension est valide pour quel fait ;
- créer de nouvelles entrées dans les dimensions (ne pas faire de mise à jour au sens d'UPDATE) ;
- ...

Méthode : Stratégies de mise à jour

- Rafraîchissement périodique
- Rafraîchissement manuel
- Rafraîchissement événementiel
- ...

II

Proposition d'architecture simplifiée pour un ETL ex nihilo, batch, non incrémental

Nous proposons un exemple d'architecture *simplifiée* pour la mise en place d'un ETL *ex nihilo*, en mode *batch*, *sans gestion du caractère incrémental.*

Cette architecture est assez générale et pourra être une base pour de nombreux cas, mais elle devra :

- être adaptée néanmoins en fonctions des spécificités propres à chaque contexte.
- être complétée (gestion incrémentale, gestion des rejets, audits...)

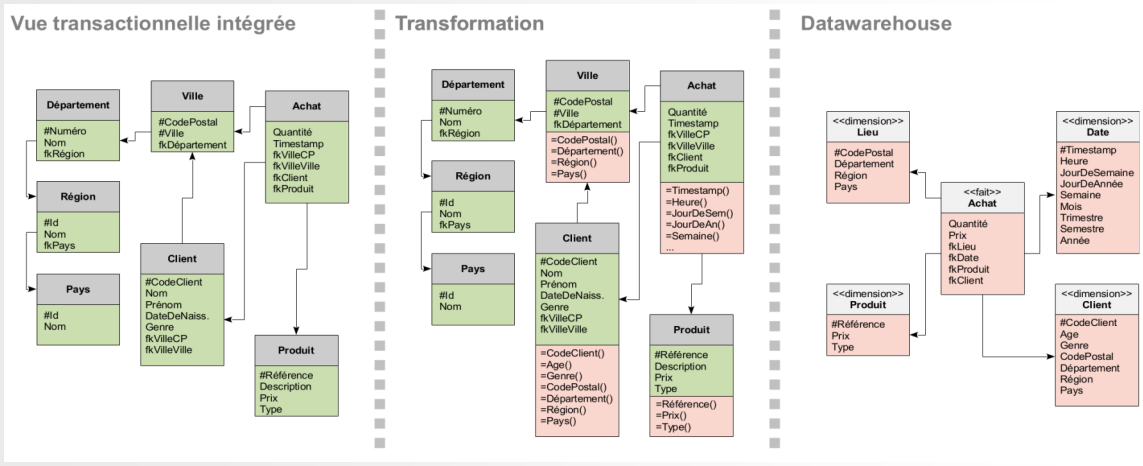
1. Architecture d'ETL à trois zones

Méthode

Nous proposons une architecture d'ETL organisée avec trois zones composées chacune d'une base de données distincte :

- *Zone d'extraction*
  Une base de données destinée à unifier les sources de données et offrir un point d'accès unique.
- *Zone de transformation*
  Une base de données destinée à traiter les sources et offrir une interface d'accès aux données transformées (API *).
- *Zone d'exploitation*
  Une base de données destinée à implémenter le data warehouse et les data marts.

Fondamental



Architecture générale d'un ETL

2. Conseils méthodologiques

Méthode : ETL multi-schéma

Dans la mesure du possible, utiliser un schéma de base de données pour chaque zone de l'ETL (BDE, BDT, DW).

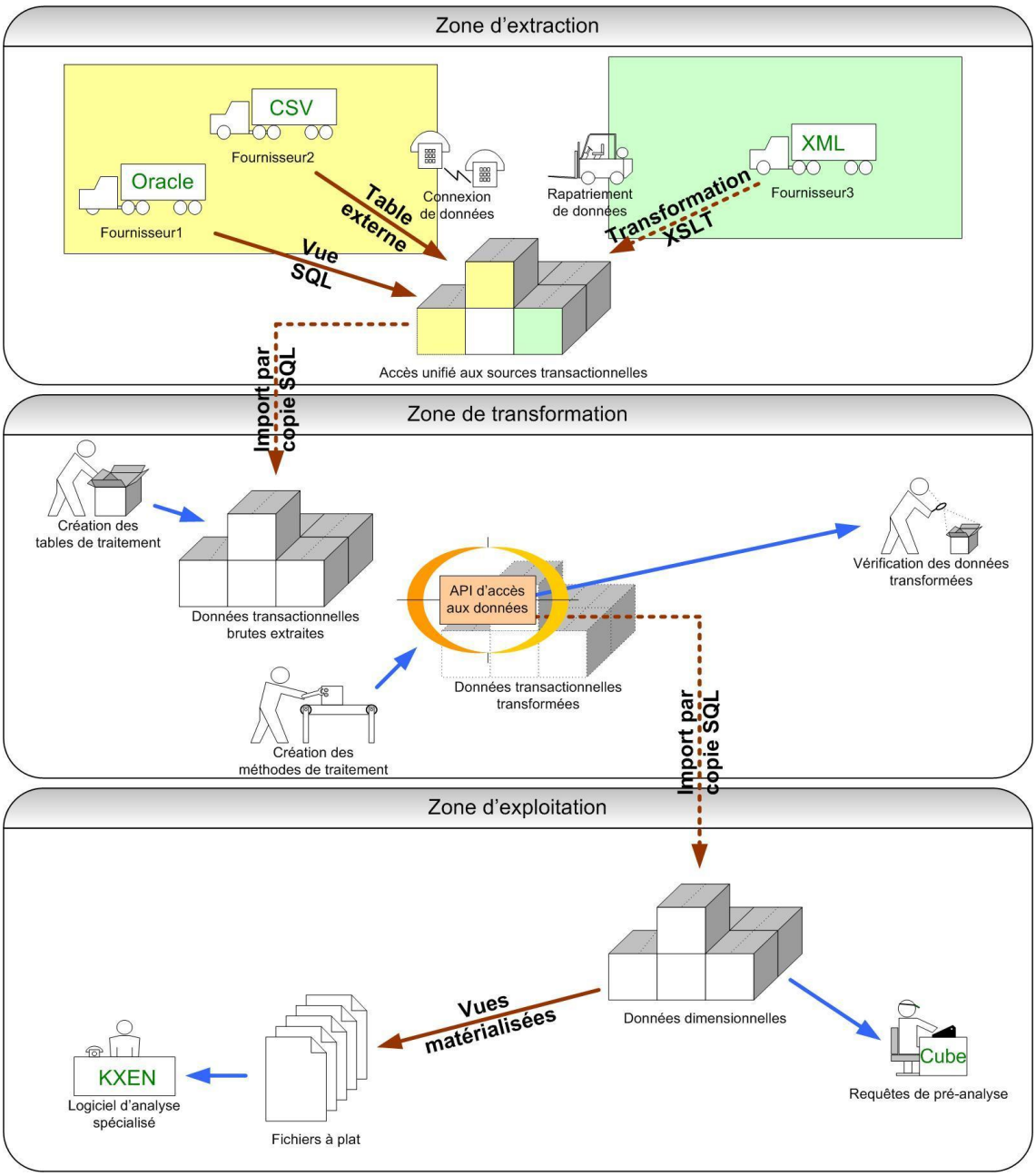
Méthode : ETL mono-schéma (contraintes de nommage)

Si tout doit être réalisé au sein d'un seul schéma, utiliser un système de pré-fixage des noms : `bde_table`, `bdt_table`, `dw_table`.

Rappel

Accès inter-schémas sous Oracle (cf. p.)

3. Résumé ETL en image



Processus ETL

4. Carte des données

Définition : Carte des données.

La carte des données (*logical data map*) est un inventaire et une mise en correspondance des données présentes dans chaque zone.

Exemple

DW				T				E			
Fact/Dim	Table	Column	Data type	Function	Table	Column	Data type	Table	Column	Data type	Source
Dim	dDate	date	YYYY-MM-DD	date()	tDate	date	char(10)	eData	date	varchar(10)	serveur.utc.fr /share/data/ data.csv
Dim	dDate	jds	1..7	jds()	tDate	date					
Dim	dDate	month	1..7	month()	tDate	date					
...											
Dim	dBook	isbn	char(13)	isbn()	tDate	isbn	varchar(255)	eCat	isbn	varchar(255)	serveur-oracle.utc schema.catalogue
Dim	dBook	author	varchar(25)	author()	tDate	author	text	eCat	auteurs	text	
...											

Exemple de cartographie des données

Complément

(Kimball, Caserta, 2004, p56-59) *

Projet Fantastic : Rappel

Cet exercice est la suite du projet Fantastic commencé dans le module *Introduction à la modélisation dimensionnelle*.

L'objectif de cet exercice est :

1. d'implémenter le data warehouse précédemment modélisé sous Oracle (serveur *sme-oracle.sme.utc*)
2. d'implémenter un processus ETL (ex nihilo, batch, non incrémental) permettant de l'alimenter avec Oracle et PL/SQL

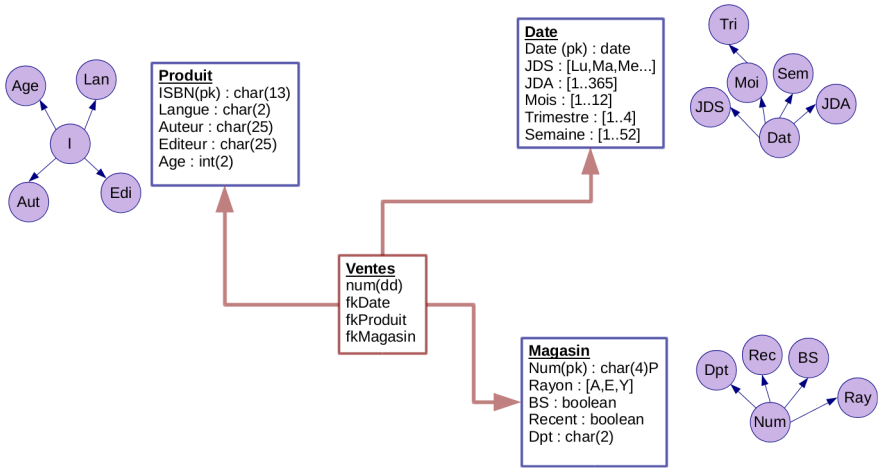
Rappel : Problème

Projet Fantastique : Problème posé (cf. p.)

Rappel : Données sources

Projet Fantastic : Données disponibles (cf. p.)

Rappel : Rappel du modèle cible



Modèle dimensionnel du data warehouse Fantastic

Implémentation simplifiée d'une zone d'extraction avec Oracle

Dans cette partie nous précisons comment implémenter pratiquement la zone E d'un processus ETL *simple*, *ex nihilo*, *batch*, *non incrémental* avec une base Oracle RO * (version 9i ou postérieure).

1. Zone E : Extraction

Définition : Base de données d'extraction

La BDE * est une BD * relationnelle destinée à implémenter la zone d'extraction d'un ETL, pour offrir un unique point d'accès à l'ensemble des sources de données.

La BD est composée de :

- tables permettant de *rapatrier* les données à importer depuis des sources externes ;
- et de vues pour se *connecter* à des sources dynamiques situées dans la même BD.

Méthode : Les fichiers CSV

Les données situées dans des fichiers CSV doivent :

1. être rapatriées sur un ou des serveurs accessibles depuis la BDE ;
2. importées dans la BDE : une table pour chaque fichier.

Il faudra automatiser le processus de copie des fichiers si les données sont susceptibles d'être mises à jour.

Remarque : Tables externes

Certains SGBD, comme Oracle, propose une alternative à l'import, grâce à un concept de table externe qui permet de lier dynamiquement une définition de table à un fichier CSV.

Méthode : Les autres fichiers : tableurs, XML…

Pour les fichiers autres que CSV, deux solutions sont à étudier :

- soit votre BDE offre une API d'accès direct à ces formats ;
- soit le fichier est transformé en CSV.

Dans le second cas, il faudra automatiser la transformation si le fichier est susceptible de mises à jour.

Méthode : Les données stockées en BD

Pour les données stockées en BD, trois solutions sont à étudier :

- si la BDE et la BD source sont sur le même SGBD, on créé simplement une vue ;
- sinon, lorsque c'est possible on établit un lien dynamique entre la BDE et les tables sources (propriétaire, ODBC ou JDBC) ;
- sinon, on fait un export de la BD source dans un fichier CSV (en gérant l'automatisation de l'export lorsque la base est vivante).

Méthode : Gestion des contraintes

Les contraintes doivent être relâchées au maximum dans la BDE pour assurer que les données sources seront toutes correctement accessibles.

On veillera à avoir correctement documenté les contraintes connues, notamment pour les données provenant de SGBD dans lesquels ces contraintes sont formalisées dans le schéma.

2. Sources de données

Méthode : Données sur le même serveur Oracle

Réaliser des vues pour accéder dynamiquement aux données sources.

Méthode : Données dynamiques en fichier CSV

Créer une *table externe* Oracle pour accéder dynamiquement à ce fichier.

Si le fichier n'est pas accessible directement depuis le serveur Oracle, procéder à une copie (automatisée par script).

Méthode : Données statiques

Pour les données ne demandant aucune mise à jour, ou des mises à jour très ponctuelles :

- faire un export depuis la source en CSV ;
- créer une table Oracle dans la BDE * ;
- importer manuellement le CSV avec SQL Developer.

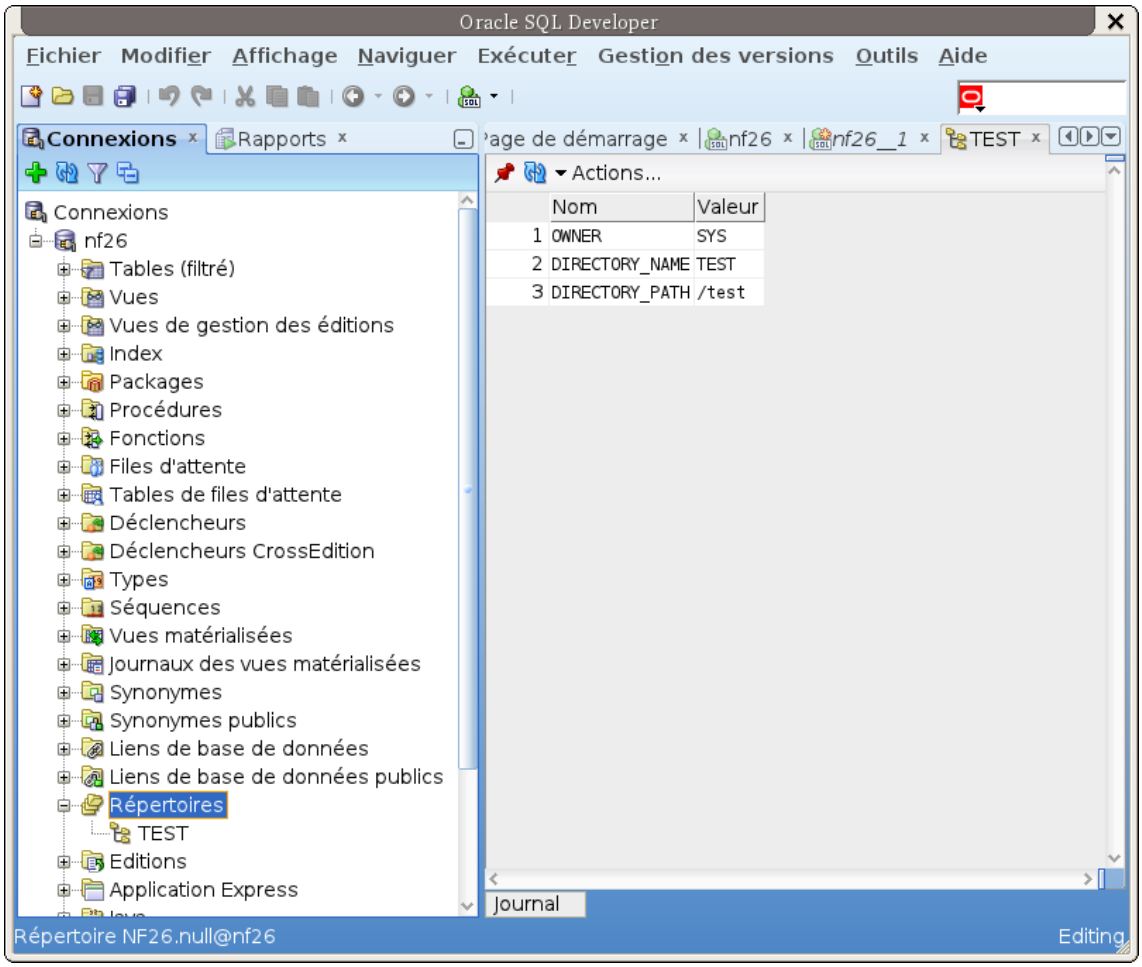
3. Tables externes sous Oracle

Définition : Table externe

Une table externe sous Oracle est une méthode d'accès sans copie à des fichiers CSV dynamiques exactement comme s'il s'agissait d'une table de la BD.

Syntaxe : Préambule : Déclaration des répertoires de travail

```
1 CREATE OR REPLACE DIRECTORY <nom du répertoire source> AS '<chemin du répertoire de la
  source>';
2 CREATE OR REPLACE DIRECTORY <nom du répertoire log> AS '<chemin du répertoire des fichiers de
  log>';
```



Exemple de répertoire de travail

Attention : Accès aux répertoire de travail

Le répertoire de la source et le fichier source doivent être accessibles en lecture pour le processus Oracle.

Le répertoire des fichiers de log doit être accessible en lecture et écriture pour le processus Oracle.

⚠ *Attention*

« All directories are created in a single namespace and are not owned by an individual schema »

http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5007.htm

Syntaxe : Création d'une table externe

```
1 CREATE TABLE <nom de la table> (
2 <déclaration des attributs avec domaine mais sans contrainte>
3 )
4 ORGANIZATION EXTERNAL
5 (TYPE ORACLE_LOADER
6 DEFAULT DIRECTORY <répertoire déclaré préalablement>
7 ACCESS PARAMETERS
8 (
9 RECORDS DELIMITED BY '<caractère de fin de ligne>'
10 SKIP <nombre de lignes à ignorer>
11 CHARACTERSET <encodage des caractères>
12 BADFILE <répertoire>:'<fichier>'
13 LOGFILE <répertoire>:'<fichier>'
14 FIELDS TERMINATED BY '<séparateur de champ>'
15 OPTIONALLY ENCLOSED BY '<séparateur de chaîne>'
16 )
17 LOCATION ('<fichier source>'))
18 REJECT LIMIT UNLIMITED;
```

⚠ *Attention : Ne pas utiliser de commentaire au sein la déclaration des paramètres d'accès*

« One important point to remember is that comments must be placed before any access parameters. If you include comments in the access parameter sections, Oracle will throw an error when you query the external table but not when you are creating it. »

https://oracleappsnotes.wordpress.com/2012/02/10/oracle-external-tables-a-few-examples/

Exemple

```
1 CREATE OR REPLACE DIRECTORY monRepertoireSrc AS '/user1c/nf26/nf26/projet/csv/';
2 CREATE OR REPLACE DIRECTORY monRepertoireLog AS '/volsme/user1x/uvs/nf26/nf26p099/test/';
3 /
```

```
1 CREATE TABLE tImport (
2 a VARCHAR(50),
3 b NUMBER(10)
4 )
5 ORGANIZATION EXTERNAL
6 (TYPE ORACLE_LOADER
7 DEFAULT DIRECTORY monRepertoireSrc
8 ACCESS PARAMETERS
9 (
10 RECORDS DELIMITED BY newline
11 SKIP 1
12 CHARACTERSET UTF8
```

```
13 BADFILE monRepertoireLog:'import.bad'
14 LOGFILE monRepertoireLog:'import.log'
15 FIELDS TERMINATED BY ';'
16 OPTIONALLY ENCLOSED BY '"'
17 )
18 LOCATION ('sources.csv'))
19 REJECT LIMIT UNLIMITED;
```

Description des paramètres

- `DEFAULT DIRECTORY` : Le répertoire où se trouvent le fichier source
- `RECORDS DELIMITED BY` : Séparateur d'enregistrements (`newline` est le caractère de fin de ligne standard du système)
- `SKIP 1` : permet d'ignorer la première ligne d'un fichier CSV lorsque celui-ci contient les entêtes de colonnes
- `CHARACTERSET` : permet de spécifier l'encodage des caractères (UTF8...)
- `BADFILE` : Fichier contenant les enregistrements rejetés à l'import (récréé à chaque exécution)
- `LOGFILE` : Fichier contenant les traces d'exécution (traces ajoutées à chaque exécution)
- `FIELDS TERMINATED BY` : Séparateur de champs (en général `;` dans un CSV)
- `OPTIONALLY ENCLOSED BY` : Séparateur de chaînes (en général `"` dans un CSV)
- `LOCATION` : Nom du fichier dans le répertoire sélectionné
- `REJECT LIMIT` : Nombre d'enregistrements pouvant être rejetés avant interruption de la requête (un entier ou UNLIMITED)

Syntaxe : Vérification de la déclaration de la table externe

```
1 DESCRIBE Timport;
2 SELECT TABLE_NAME, TYPE_NAME, DEFAULT_DIRECTORY_NAME FROM
  USER_EXTERNAL_TABLES;
```

Attention : Accès

L'accès à la source externe CSV par Oracle ne se fait en fait qu'à la première interrogation (`SELECT`), donc il est nécessaire d'exécuter un appel à la table pour valider cet accès (seule une vérification syntaxique est faite au moment du `CREATE TABLE`, le fichier CSV peut même ne pas exister).

En pratique si les données comportent des erreurs, les problèmes se déclareront à ce moment là. On consultera le fichiers de log et des enregistrements rejetés pour le savoir.

```
1 SELECT * FROM Timport;
```

Attention : Gros fichiers

Si les volumes de donnée sont importants (s'il y a beaucoup de lignes), privilégiez l'usage de la clause ROWNUM pour éviter de faire transiter des méga-octets de données entre le serveur et le client. Les délais de réaction en seront améliorés.

Vous pouvez aussi utiliser des SUM, MIN, MAX, etc. pour être sûr que toutes les lignes et colonnes sont correctement lues.

Conseil : Tout recompter

Une fois la procédure terminée effectuer un `count(*)` pour déterminer si l'import a bien traité toutes les lignes du fichier source.

Remarque : NUMBER(X)

Pour spécifier des entiers dans les tables externes, utiliser `NUMBER(X)`, avec le `(X)` obligatoire.

Remarque : Caractère de fin de ligne

`RECORDS DELIMITED BY newline` signifie que le caractère standard du système est utilisé comme caractère de fin de ligne.

Or :

- sous Unix le caractère de fin de ligne est "\n"
- sous Windows il est "\r\n"

Donc, si un fichier CSV est encodé sous un système et lu sous un autre, il y aura un problème. Par exemple si le fichier est encodé sous Windows il aura "\r\n" à la fin de chaque ligne, et s'il est lu sous Linux, Oracle cherchera uniquement un "\n", d'où un "\r" résiduel sera considéré comme faisant partie du dernier champ.

La solution la plus robuste est de spécifier *en dur* le caractère de fin de ligne :

- Si le fichier source a été encodé sous Unix : RECORDS DELIMITED BY '\n'
- Si le fichier source a été encodé sous Windows : RECORDS DELIMITED BY '\r\n'

Conseil

Penser à gérer les fichiers de rejet et de log. Le plus simple est de les supprimer après une exécution incorrecte.

- Les fichiers de rejet ne sont pas créés si l'exécution est correcte (et donc un éventuel fichier existant n'est pas modifiée par une exécution correcte)
- Les fichiers de log grossissent à chaque exécution

4. Exemple de chargement de données depuis un CSV par une table externe

Il est possible d'utiliser les tables externes pour charger des données issues d'un fichier CSV dans une table existante.

Exemple

Soit la table `tTypeDefaut` à charger avec un fichier `tTypdeDefaut.txt`.

```
1 tTypeDefaut (#pkTypeDefaut:number(4), libelle:varchar(50), fkFamilleDefaut:char(1)=>tFamilleDefaut)
```

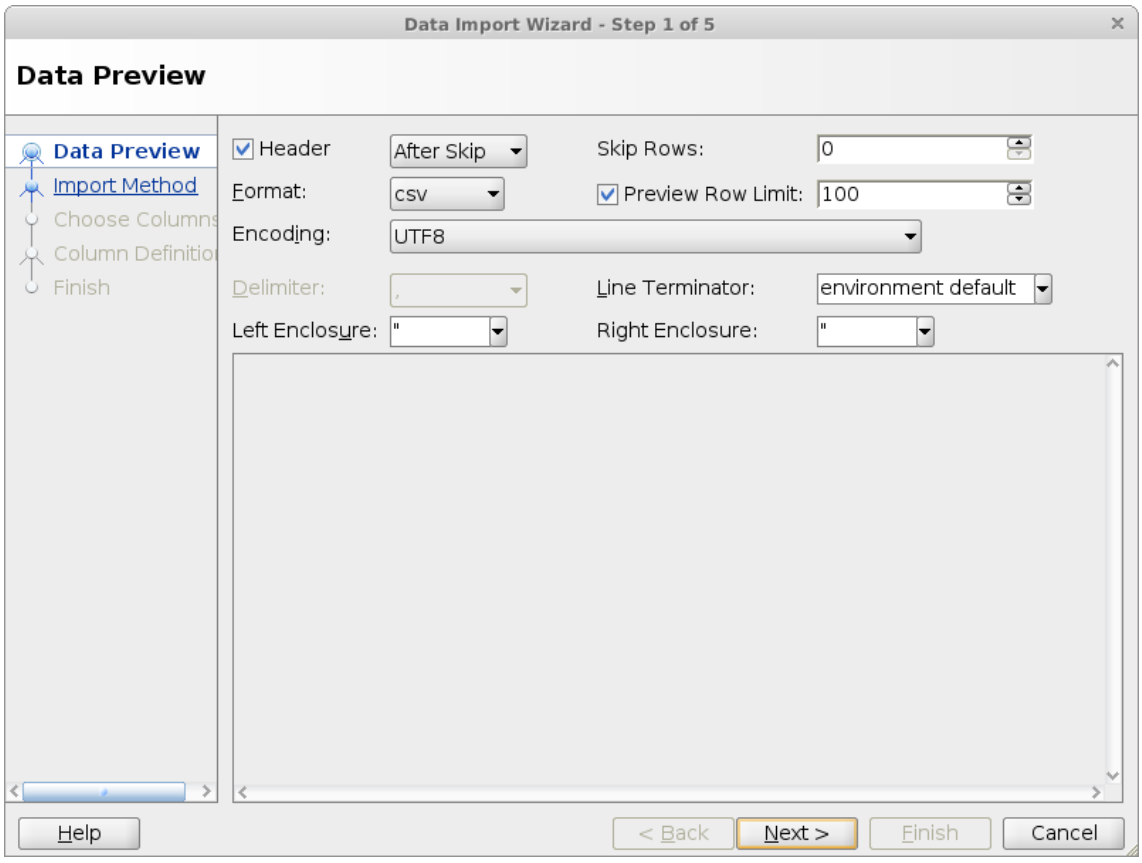
On peut utiliser une table externe :

```
CREATE TABLE tTypeDefautLoad (
pkTypeDefaut number(4),
libelle varchar(50),
fkFamilleDefaut char(1)
)
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
DEFAULT DIRECTORY srcDir
ACCESS PARAMETERS
(
RECORDS DELIMITED BY NEWLINE
BADFILE logDir:'import.bad'
LOGFILE logDir:'import.log'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"')
LOCATION ('tTypeDefaut.txt'))
REJECT LIMIT UNLIMITED;
```

```
INSERT INTO tTypeDefaut
SELECT * FROM tTypeDefautLoad;
```

5. Insertion CSV manuelle avec SQL Developer

Effectuer un clic droit sur le dossier `tables` à gauche dans SQL Developer et choisir `Import data` pour accéder à un assistant de chargement manuel.



Import manuel de données avec Oracle SQL Developer

V

Exercice : Projet Fantastic : Mise en place de la zone d'extraction

L'objectif est d'abord de créer la BDE. Les noms des tables et vues seront préfixés :

- `f_` dans le schéma `bde` (f pour projet Fantastique) ;
- ou `f_bde_` si vous ne disposez que d'un seul schéma pour toutes vos BD.

Question 1

Créez une vue `f_bde_catalogue` pour la table de la base Oracle "catalogue".

Indices :

Enregistrez l'instruction de création de votre vue dans un fichier `f_bde_catalogue.sql`.

- Créez un fichier `f_bde.sql` qui va appeler toutes vos instructions de création d'objets dans la BDE.
  Sa première ligne est donc : `@f_bde_catalogue.sql`
- On pourra éventuellement créer un fichier `f.sql` qui appellera les scripts `f_bde.sql`, `f_bdt.sql` et `f_dw.sql`

Gestion de fichiers SQL (cf. p.42)

Utiliser l'instruction `CREATE OR REPLACE VIEW` pour la création de la vue, pour permettre la recréation de la vue par le script `f_bde.sql`.

Question 2

Créez une table externe pour chacun des fichiers `marketing.ods` et `departementsInsee2003.txt`.

Indices :

- Faites un export CSV du fichier `marketing.ods` vers le fichier `marketing.csv`
- Copiez le, ainsi que `departementsInsee2003.txt`, dans un dossier `data` de votre compte sur le serveur `sme-oracle.sme.utc`
- Ouvrez l'accès à ce dossier en lecture (`chmod 755`)
- Créez un dossier `tmp` ouvert en lecture et écriture (`chmod 777`)
- Créez les objets `DIRECTORY` permettant de pointer sur les répertoires `data` et `tmp`.
- Créez une table externe pour `departementsInsee2003.txt`, en envoyant les fichiers de rejet `departementsInsee2003.txt.bad` et de log `departementsInsee2003.txt.log` dans votre dossier `tmp`.
- Testez votre table externe : `SELECT * FROM ...`
- Vérifiez que tout s'est bien passé en examinant les fichiers de rejet et de log
- De la même façon, créez une table externe pour `marketing.csv` et vérifier l'accès aux données

Pensez que les objets `DIRECTORY` sont partagés au niveau de toute l'instance et ne sont pas spécifiques à un schéma.

Donc si deux `users` créent un même `DIRECTORY` nommé `tmp`, il y aura un conflit (la seconde création écrasera la première).

Pour accéder à un répertoire `d1` situé dans un répertoire `d0`, `d0` doit être accessible en exécution (`chmod 711`).

Pour lire les fichiers `.log` et `.bad` :

```
more ~/tmp/import.log

more ~/tmp/import.bad
```

Vous pouvez vider les fichiers `.log` régulièrement pour en faciliter la lecture.

```
echo > ~/tmp/import.log
```

Vous pouvez supprimer les fichiers `.bad` après avoir traité les causes d'un rejet (si une exécution ne génère pas de rejet elle ne crée pas de fichier de rejet, et ne modifie donc pas un éventuel fichier existant).

```
rm ~/tmp/import.bad
```

Question 3

Créez une table externe pour le fichier `data.csv`

Indices :

- Créez un nouvel objet `DIRECTORY` permettant de pointer sur le répertoire `/home/nf26/data`

Attention le fichier `data.csv` est très volumineux, aussi ne faites pas de `SELECT *` dessus, sous peine d'attendre longtemps la fin de l'exécution (le serveur devant renvoyer plusieurs Mo de données).

Faites des `SELECT` partiels avec la clause `ROWNUM` pour limiter les données à rapatrier et des select avec opérations de regroupement (min ou max typiquement).

```
1 SELECT ... FROM ... WHERE rownum<=N;
```

Notez qu'en cas d'erreur à l'import portant sur toutes les lignes, le fichier de log risque de devenir très volumineux, pouvant conduire à la saturation de votre compte. Videz le fichier de log après avoir généré une telle erreur.

Pour vérifier la taille du log :

```
ls -l ~/tmp/import.log
```

Pour tester toutes les lignes : `SELECT count(*), min(...), max(...), max(...) ...`

VI Implémentation simplifiée d'une zone de transformation avec Oracle

Dans cette partie nous précisons comment implémenter pratiquement la zone T d'un processus ETL *simple*, *ex nihilo*, *batch*, *non incrémental* avec une base Oracle RO * (version 9i ou postérieure).

1. Zone T : Transformation

Définition : Base de données de transformation

La BDT * est une BD * relationnelle ou relationnel-objet destinée à implémenter la zone de transformation d'un ETL.

- Elle reçoit une copie des données issue de la zone E.
- Elle permet les transformations des données.
- Elle offre un accès aux données transformées via une API permettant une copie vers la zone L.

Définition : API de la BDT

L'API de la BDT est un ensemble de fonctions - ou méthodes si l'on dispose d'un SGBDRO * qui les autorise - qui permet d'accéder aux données de façon stable (principe d'encapsulation).

L'API de la BDT permet de rendre le chargement du data warehouse moins dépendant aux variations dans les sources de données.

Le principe proposé est le suivant :

- Créer une fonction pour chaque attribut existant dans la data warehouse.
- Appeler ces fonctions lors du chargement du data warehouse, au lieu d'appeler directement les attributs des tables de la BDT.
- Chaque fonction est en charge d'appeler les attributs de la BDT et de faire les traitements nécessaires pour fournir la valeur souhaitée.

Exemple : Exemple d'appel à l'API de la BDT en RO

```
1 SELECT t.fpk(), t.fdate(), t.fjds(), t.fmois(), t.ftrimestre()
2 FROM t_date t
```

Fondamental : Structure de la zone T

La zone T est donc composée :

- d'une BDT dont le schéma correspond à un schéma transactionnel représentant l'intégration des différentes sources (il est similaire à celui de la zone E).
- d'une API permettant d'accéder aux données (et d'exécuter des transformation)

Attention

On notera que la zone de transformation copie depuis la zone d'extraction les données en l'état, sans transformation. Donc elle contient des données encore "sales".

En revanche, les données qui sortent de cette zone pour aller vers la zone d'analyse - les données disponibles via l'API - sont traitées et donc "propres".

Méthode : Transformation simple

Les transformations simples, typiquement qui ne nécessitent qu'un seul enregistrement en entrée, seront effectuées directement et dynamiquement par les fonctions de l'API (ou des fonctions appelées par celles-ci).

Méthode : Transformations complexes

Les transformations plus complexes devront être réalisées par des procédures exécutées en batch après le chargement de la BDT.

Elle produiront des données complémentaires stockées dans la BDT.

Méthode : Vues

Les vues doivent être utilisées pour unifier (UNION) ou joindre (JOIN) les différentes tables physiques qui représentent une même donnée logique.

Les vue matérialisées peuvent être utilisées et rafraîchies à chaque mise à jour de la zone T.

Méthode : Gestion des contraintes

Les contraintes de la zone T doivent être compatibles avec les données sources afin de :

- laisser passer 100% des données depuis la zone E (sans contrainte) vers la zone T (avec contrainte) ;
- s'assurer le maximum d'information sur la nature de données

Si les contraintes sont trop relâchées, il faudra faire des vérifications inutiles pour contrôler des données, qui en fait avaient déjà les propriétés souhaitées.

Si les contraintes sont trop fortes, toutes les données ne passeront pas.

Complément

Vue matérialisée sous Oracle 9i (cf. p.)

2. Implémentation de la zone T en RO

Méthode

- On crée une table pour chaque vue, table externe et table classique de la zone d'extraction.
  Ces tables sont créées selon la syntaxe SQL3 du modèle relationnel-objet (afin de pouvoir accepter des méthodes).
  On a donc un attribut disponible dans la zone T pour chaque attribut de la zone E.
- On déclare une méthode pour chaque attribut que l'on souhaite exporter dans le modèle dimensionnel.
  Cette méthode permettra de réaliser dynamiquement les transformations et vérifications adéquates.
  On a donc une méthode disponible dans la zone T pour chaque attribut voulu dans le DW.

Rappel : Transformation simples et complexes

Pour les méthodes simples ne portant que sur un enregistrement à la fois, la méthode est attachée à la table correspondante.

Pour les transformations complexes nécessitant un script préalable, la méthode est associée à la table où est stockée le résultat de ce script.

Des vues peuvent être créées pour unifier l'accès aux tables.

Rappel : Contraintes

Pour chaque hypothèse de "propreté" des données sources on pose une contrainte associée.

- Par exemple si une donnée doit être une clé primaire et que l'on pense que les sources sont correctes de ce point de vue, on ajoute la clause PRIMARY KEY.
- Par contre il ne faut pas ajouter les contraintes lorsque l'on sait que les données sources sont "sales", sans quoi ces données seront refusées au chargement et ne pourront jamais être nettoyées.
- Dans le doute, il est parfois utile de donner des tailles de champs plus grandes que celle attendues (par exemple une chaîne de 50 caractères au lieu de 20) ou bien des types plus permissifs (une chaîne au lieu d'une date) afin de ne pas bloquer ou tronquer d'enregistrement.

Le relâchement des contraintes demandera un travail plus important d'implémentation des méthodes.

Méthode : Implémentation des méthodes

- Lorsque les données sources sont "propres" et qu'elles sont copiées telle qu'elle dans la cible, la méthode associée se contente d'un `return` de l'attribut correspondant.
- Lorsqu'un traitement est nécessaire, il est implémenté au sein de la méthode.

Fondamental

Pour être "propre" une donnée doit déjà respecter les contraintes souhaitées dans la zone T (condition nécessaire, non suffisante).

Dans tous les autres cas, la méthode doit effectuer des vérifications et traitements.

Attention : Collecte des statistiques

Il faut re-calculer les statistiques nécessaires à l'optimisation du moteur de requêtes (ANALYSE) comme après tout ajout significatif de données.

Pensez également à rafraîchir les vues matérialisées, puis à exécuter les collectes de statistiques sur ces vues.

Complément

Plans d'exécution sous Oracle 9i (cf. p.)

3. Désactivation et réactivation de contraintes

Rappel

Le chargement dans la base dimensionnelle, si les méthodes de transformation ont été correctement écrites, ne comporte que des données valides. De plus ce chargement va impliquer un nombre très important de données. Il est donc souhaitable de désactiver les contraintes sur le modèle dimensionnel pendant le temps de chargement, afin d'accélérer cette procédure.

Notons que si les contraintes ne sont pas désactivées :

- à chaque ajout d'une ligne le moteur de la base va devoir vérifier que cette ligne respecte les contraintes ;
- de plus si les données ne sont pas chargées exactement dans le bon ordre, des contraintes de type intégrité référentielle peuvent être temporairement non validées.

Précisons enfin qu'une fois le chargement terminé les contraintes seront réactivées afin de vérifier que les méthodes de transformation ont fait correctement leur travail et que les données respectent effectivement les contraintes du modèle dimensionnel. Si les nouvelles données ne respectent pas les contraintes, ces dernières ne pourront être réactivées tant que les erreurs n'auront pas été corrigées.

Méthode : Préalable à la réactivation

Oracle fournit un script (`utlexcpt.sql`) pour la création d'une table qui va servir à récupérer les éventuelles erreurs détectées suite à la réactivation des contraintes.

```
1 -- utlexcpt.sql
2 create table exceptions(row_id rowid,
3                         owner varchar2(30),
4                         table_name varchar2(30),
5                 constraint varchar2(30));
```

Syntaxe : Désactivation de contraintes

```
1 ALTER TABLE nom_table DISABLE CONSTRAINT nom_contrainte;
```

Remarque : Déclaration des contraintes à la création

On notera qu'il est important pour pouvoir aisément désactiver les contraintes de les avoir explicitement nommées lors de la création des tables.

Syntaxe : Réactivation des contraintes

```
1 ALTER TABLE nom_table ENABLE CONSTRAINT nom_contrainte
2
3  EXCEPTIONS INTO exceptions ;
```

Méthode : Erreurs de réactivation des contraintes

Si les contraintes ne peuvent être réactivées du fait que certaines données ne sont plus conformes, les enregistrements en cause seront référencés (par leur *rowid*) dans la table `exceptions` créée par le script `utlexcpt.sql`. Pour retrouver ces enregistrements, exécuter une requête de sélection dans cette table.

```
1 SELECT * FROM nom_table WHERE rowid IN (SELECT row_id FROM exceptions);
```

Une fois les erreurs corrigées, l'opération de réactivation des contraintes peut être renouvelée.

Complément : Désactivation des contraintes et suppression des index

Un contexte qui nécessite la désactivation des contraintes pour améliorer des performances de chargement nécessitera également la suppression des index, également gourmands en ressources lors de la création ou mise à jour des données. Ces index seront recréés une fois le chargement terminé.

4. Processus de chargement BDE->BDT

Attention : Performance du chargement

Afin d'améliorer les performances au chargement dans une zone, on désactive les contraintes et on supprime les index préalablement au chargement, et on les réactive et recrée postérieurement.

Méthode : Chargement BDE->BDT

1. Désactivation des contraintes de la BDT
2. Suppression des index de la BDT
3. Copie des données de la BDE vers la BDT
4. Recréation des index
5. Réactivation des contraintes
6. Vérification que la réactivation des contraintes n'a pas rejeté de données (100% des données sont passées de la BDE à la BDT)
7. Exécution des procédures de pré-traitement

8. Actualisation des vues matérialisées
9. Signalisation de la disponibilité de la zone T

⚠ *Attention : Passage obligé*

Le passage d'une zone à l'autre doit toujours laisser passer *toutes* les données.

Lors du passage de la zone d'extraction à la zone de transformation, les contraintes qui bloquent doivent être levées pour laisser passer les données et lors du passage de la zone de transformation à la zone d'exploitation, les méthodes doivent gérer tous les cas de figure problématiques.

VII

Exercice : Projet Fantastic : Mise en place de la zone de traitement

L'objectif est à présent de créer la BDT en RO. Les noms des tables et vues seront préfixés :

- `f_` dans le schéma `bdt`
- ou `f_bdt_` si vous ne disposez que d'un seul schéma pour toutes vos BD.

Question 1

Créez une table RO `f_bdt_catalogue` avec les attributs de la vue catalogue `f_bde_catalogue` et une méthode pour chaque attribut de la dimension `produit`..

Indices :

Les méthodes ne sont pas implémentées pour le moment.

Pensez à déclarer les contraintes et index explicitement pour pouvoir les désactiver plus tard, avant les chargements massifs.

Question 2

Créez une table RO `f_bdt_magasin` destinée à recevoir la jointure des tables associées à `marketing.ods` et `departementsInsee2003.txt`.

Indices :

Les départements ne sont pas identifiés exactement de la même façon dans les deux tables `f_bde_dpt` et `f_bde_marketing`, il n'est donc pas possible des les joindre directement.

Une solution consiste à créer deux vues `f_bdt_dpt` et `f_bdt_marketing` qui vont permettre d'ajuster les valeurs avant la jointure. Par exemple :

- La vue `f_bdt_dpt` renvoie directement les valeurs de `f_bde_dpt` (on peut éventuellement s'en passer)
- La vue `f_bdt_marketing` renvoie des valeurs corrigées qui peuvent être jointes à `f_bdt_dpt`

D'autres solutions auraient été :

- De gérer la modification directement dans la requête INSERT vers `f_bdt_magasin`
- De faire l'insertion dans `f_bdt_magasin` via un script PL/SQL (qui traite les cas problématiques)
- De copier les données de `f_bde_marketing` dans une table `f_bdt_marketing`, d'exécuter un script corrigeant les données, puis de faire le INSERT
- ...

Pour traiter le problème des numéros de département qui sont de type 1, 2 ... au lieu de 01, 02... on peut utiliser un CASE dans un SELECT :

```
1 SELECT
2 CASE WHEN TO_NUMBER(dpt)<10 THEN '0'||TO_NUMBER(dpt) ELSE dpt END AS dpt,
3 ...
4 FROM f_bde_marketing;
```

Question 3

Créez une table RO `f_bdt_date` avec un seul attribut qui recevra les valeurs d'un `select distinct dat ...` depuis la table externe permettant d'accéder à `data.csv`.

Question 4

Créez une table RO `f_bdt_vente` pour alimenter la table des faits.

Question 5

Écrivez la procédure d'import BDE->BDT en suivant bien les étapes du processus de chargement.

Indices :

Désactivez vos contraintes et index avant le chargement.

Vérifiez votre import (pas de rejet, vérification du nombre de lignes...)

Penser à valider votre transaction (`COMMIT`).

Question 6

Implémentez une première version des méthodes qui ne fera aucun traitement pour le moment : la méthode retourne soit un attribut sans transformation, soit une constante si ce n'est pas possible.

VIII Implémentation simplifiée d'un data warehouse avec Oracle

Dans cette partie nous précisons comment implémenter pratiquement la zone L d'un processus ETL *simple*, *ex nihilo*, *batch*, *non incrémental* avec une base Oracle RO * (version 9i ou postérieure).

1. Zone L : Loading

Définition : Base de données de chargement ou data warehouse

La zone de chargement est en fait la BD en étoile ou en flocon qui implémente le data warehouse et les data marts.

Elle reçoit une copie des données sous leur forme transformée (depuis la zone T) disponible pour l'exploitation.

Rappel

Cette BD dimensionnelle peut être :

- directement exploitée pour effectuer des requêtes ;
- ou utilisée pour effectuer des exports vers des logiciels d'analyse spécialisés.

2. Implémentation du data warehouse en R

L'implémentation d'un modèle dimensionnel en base relationnelle ne diffère pas dans sa première phase de celle d'un modèle transactionnel.

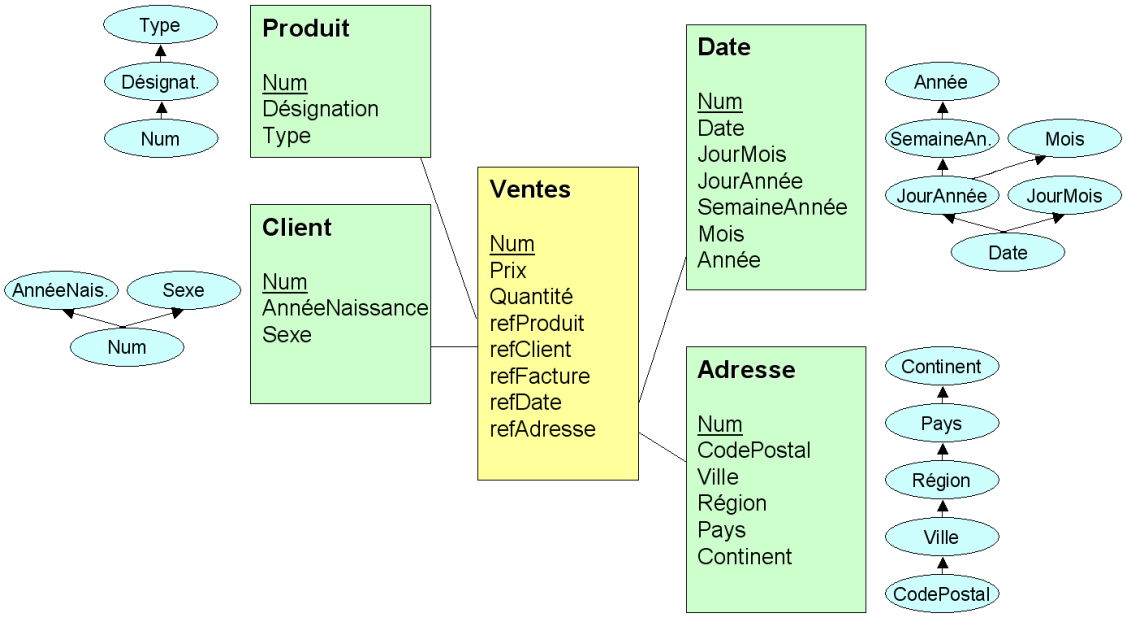
On privilégiera simplement un nommage explicite des contraintes et index pour en faciliter la manipulation ultérieure (désactivation, optimisation...).

Conseil : Déclaration explicite des contraintes et index

On utilisera typiquement la syntaxe ci-après pour déclarer une relation dotée d'une clé primaire dont on connaît le nom et pour laquelle on connaît l'index sur lequel elle se base. Notons qu'une déclaration classique de clé primaire aurait conduit à créer un index automatiquement, donc plus difficile à gérer (avec un nom généré par Oracle).

```
1 CREATE TABLE table1 (pknum number, ...);
2 CREATE UNIQUE INDEX idx_table1_pknum   ON table1 (pknum);
3 ALTER TABLE table1 ADD CONSTRAINT cstr_table1_pknum PRIMARY KEY (pknum) ;
```

Exemple : Implémentation d'un modèle dimensionnel



Modèle dimensionnel

```
1 CREATE TABLE t_produit (
2   pk_num number,
3   a_designation varchar(50),
4   a_type char(3)
5 );
6
7 CREATE UNIQUE INDEX idx_produit_num
8   ON t_produit (pk_num);
9 ALTER TABLE t_produit
10   ADD CONSTRAINT cstr_produit_num PRIMARY KEY (pk_num)
11   ADD CONSTRAINT cstr_produit_type CHECK (a_type in ('CD', 'DVD'));
12
13 ...
```

3. Processus de chargement BDT->DW

Méthode : Chargement BDT->DW

1. Désactivation des contraintes du DW * et des DM *
2. Suppression des index du DW et des DM
3. Chargement chaque dimension du data warehouse via l'API de la zone T
4. Chargement la table des faits du data warehouse via l'API de la zone T
5. Recréation des index du DW
6. Réactivation des contraintes du DW
7. Vérification que la réactivation des contraintes n'a pas rejeté de données (100% des données sont passées de la BDT au DW)
8. Extraction des data marts
9. Recréation des index des DM
10. Réactivation des contraintes des DM
11. Extraction des fichiers plats destinés aux applications d'exploitation
12. Signalisation de la disponibilité du DW.

Rappel : Performance du chargement

Afin d'améliorer les performances au chargement dans une zone, on désactive les contraintes et on supprime les index préalablement au chargement, et on les réactive et recrée postérieurement.

Rappel : Passage obligé

Le passage d'une zone à l'autre doit toujours laisser passer *toutes* les données.

IX

Exercice : Projet Fantastic : Mise en place de la zone d'exploitation

L'objectif est maintenant de créer le DW en R. Les noms des tables et vues seront préfixés :

- `f_` dans le schéma `dw`
- ou `f_dw_` si vous ne disposez que d'un seul schéma pour toutes vos BD.

Question 1

Réalisez l'implémentation SQL LDD de votre modèle dimensionnel de DW.

Question 2

Écrivez et testez la procédure d'import BDT->DW.

Question 3

Documentez votre processus ETL complet en effectuant une carte des données.

Exercice : Projet Fantastic : Implémentation des transformations

X

Afin de finaliser l'ETL, il est à présent nécessaire de terminer l'implémentation des méthodes pour renvoyer les valeurs recherchées.

Question 1

Implémentez les méthodes pour la table des faits.

Indice :

Pour le traitement de la conversion de la date de type `varchar` en type `date` on utilisera la commande `TO_DATE`.

Question 2

Implémentez les méthodes effectuant un simple `return` pour la dimension "magasin" (aucun traitement).

Question 3

Implémentez les méthodes de la dimension "date".

Indice :

- Utilisez la fonction `TO_DATE` pour obtenir une valeur de type date pour la méthode date()
- Puis utilisez la fonction `TO_CHAR` sur cette méthode pour obtenir les autres attributs

Fonctions SQL à connaître (cf. p.42)

Question 4

Implémentez des méthodes effectuant des tests de vérification de format et lorsque c'est nécessaire un reformatage pour la dimension "produit".

Question 5

Exécutez la procédure d'import BDT->DW.

Faites toutes les vérifications nécessaires.

Rappels

XI

1. Rappels Oracle pour l'ETL

1.1. Création de vues

Définition : Vue

Une vue est une définition logique d'une relation, sans stockage de données, obtenue par interrogation d'une ou plusieurs tables de la BD *. Une vue peut donc être perçue comme une fenêtre dynamique sur les données, ou encore une requête stockée (mais dont seule la définition est stockée, pas le résultat, qui reste calculé dynamiquement).

Une vue permet d'implémenter le concept de schéma externe d'un modèle conceptuel.

Synonymes : Relation dérivée, Table virtuelle calculée

Syntaxe

```
1 CREATE VIEW <nom de vue> <nom des colonnes>
2 AS <spécification de question>
```

La spécification d'une question se fait en utilisant le LMD *.

Le nombre de colonnes nommées doit être égal au nombre de colonnes renvoyées par la question spécifiée. Le nom des colonnes est optionnel, s'il n'est pas spécifié, c'est le nom des colonnes telle qu'elles sont renvoyées par la question, qui sera utilisé.

Exemple

```
1 CREATE VIEW Employe (Id, Nom)
2 AS
3   SELECT N°SS, Nom
4   FROM Personne
```

La vue Employe est ici une projection de la relation Personne sur les attributs N°SS et Nom, renommés respectivement Id et Nom.

Remarque : Vue en lecture et vue en écriture

Une vue est toujours disponible en lecture, à condition que l'utilisateur ait les droits spécifiés grâce au LCD *. Une vue peut également être disponible en écriture dans certains cas, que l'on peut restreindre aux cas où la question ne porte que sur une seule table (même si dans certains cas, il est possible de modifier une vue issue de plusieurs tables).

Dans le cas où une vue est destinée à être utilisée pour modifier des données, il est possible d'ajouter la clause "WITH CHECK OPTION" après la spécification de question, pour préciser que les données modifiées ou ajoutées doivent effectivement appartenir à la vue.

Remarque : Vue sur une vue

Une vue peut avoir comme source une autre vue.

Rappel : Vues et héritage

Les vues sont particulièrement utiles pour restituer les relations d'héritage perdues lors de la transformation MCD * vers MLD *.

1.2. Structure d'un bloc PL/SQL

Syntaxe

```
1 DECLARE ou IS
2    Variables, curseurs, etc.
3 BEGIN
4    Instructions SQL et PL/SQL
5 EXCEPTION
6    Gestion d'erreur.
7 END;
8 /
```

Attention : /

Un bloc PL/SQL est terminé par un ; comme une instruction SQL.

Par ailleurs, dans les environnements d'exécution Oracle (comme SQL Developer), il est nécessaire de séparer les blocs par un "/" (sur une nouvelle ligne).

Une bonne habitude est donc de terminer les blocs PL/SQL par des "/".

Complément

http://stackoverflow.com/questions/3024418/two-plsql-statements-with-begin-and-end-run-fine-seperately-but-not-together

1.3. Blocs PL/SQL : Procédure, fonction, bloc anonyme

Syntaxe : Procédure

```
1 CREATE OR REPLACE PROCEDURE nom_proc
2 IS
3    ...
4 BEGIN
5    ...
6 [EXCEPTION]
7    ...
8 END ;
```

Exemple : Procédure

```
1 CREATE OR REPLACE PROCEDURE pHello (who VARCHAR2)
2 IS
3 BEGIN
4   DBMS_OUTPUT.PUT_LINE('Hello ' || who);
5 END;
6 /
```

Syntaxe : Fonction

```
1 CREATE OR REPLACE FUNCTION nom_func
2   RETURN type_retourné
3 IS
4   ...
5 BEGIN
6   ...
7   RETURN valeur;
8 [EXCEPTION]
9   ...
10 END ;
```

Exemple : Fonction

```
1 CREATE OR REPLACE FUNCTION fDateDuJour RETURN date
2 IS
3   vDate date;
4 BEGIN
5   SELECT SYSDATE INTO vDate FROM DUAL;
6   RETURN vDate;
7 END;
8 /
```

Attention

Le type de retourné par une fonction ne doit pas spécifier de taille :

- `RETURN varchar`
- et *non* `RETURN varchar(10)`

http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm

Syntaxe : Anonyme

```
1 [DECLARE]
2   ...
3 BEGIN
4   ...
5 [EXCEPTION]
6   ...
7 END ;
```

Exemple : Script anonyme

```
1 SET SERVEROUTPUT ON;
2 BEGIN
3   pHello('World');
4   DBMS_OUTPUT.PUT_LINE(fDateDuJour);
5 END;
6 /
```

1.4. Exécution de fichiers SQL

Méthode

Pour enregistrer un script SQL ou PL/SQL écrit dans *Oracle SQL Developer* sous la forme d'un fichier utiliser la fonction `file > save as`.

Méthode

Pour exécuter un fichier SQL ou PL/SQL utiliser la commande `@fichier.sql`

Exemple : script.sql

```
1 @file1.sql
2 @file2.sql
3 @file3.sql
```

1.5. Fonctions SQL

Rappel

Par opposition aux fonctions de calcul SQL qui s'appliquent sur toute la table pour réaliser des agrégats (en ne renvoyant qu'une seule valeur par regroupement), les fonctions "mono-ligne" sont des fonctions au sens classique, qui s'appliquent à une ou plusieurs valeurs et renvoient une valeur en retour.

Les fonctions "mono-ligne" :

- Manipulent des éléments de données
- Acceptent des arguments en entrée et retournent des valeurs en sortie
- Agissent sur chaque ligne
- Retournent un seul résultat par ligne
- Peuvent modifier les types de données

Exemple

- Traitement de chaîne
    - CONCAT, SUBSTR, LENGTH, INSRT, LPAD, TRIM
    - LOWER, UPPER, INITCAP
- Traitement de date
    - MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY

    - `SELECT sysdate FROM dual`
    - Opérations mathématiques sur les dates : `SELECT sysdate + 10 FROM dual`
- Traitement numérique
    - ROUND, TRUNC
    - FLOOR, CEIL
    - MOD
- Conversion
    - Conversion implicite
    - Conversion explicite : TO_DATE, TO_NUMBER, TO_CHAR
- Générales
    - NVL (par exemple NVL(X,0) renvoie 0 si X vaut Null)
    - CASE WHEN condition1 THEN valeur1 WHEN condition2 THEN valeur2 ELSE valeur3 END
    - Imbrication de fonctions : F3(F2(F1(col,arg1),arg2),arg3)

Méthode

Les fonctions mono-ligne sont utilisées pour :

- Transformer les données
- Formater des dates et des nombres pour l'affichage
- Convertir des types de données de colonnes
- ...

Exemple : Extraction de chaîne

La fonction `SUBSTR(X, A, B)` renvoie les B caractères à partir du caractère A dans la chaîne X.

Complément

- *Fonctions SQL*
- Vous pouvez consulter Oracle : SQL *, page 9 à 12, pour avoir une description plus détaillée des fonctions disponibles sous Oracle.

Rappel : BD "Gestion des intervenants" : Schéma relationnel

```
tIntervenant (#pknom:varchar, prenom:varchar, poste:integer)
tCours (#pkannee:2000..2100, #pknum:integer, titre:varchar, type:C|TD|TP, fkintervenant=>tIntervenant, debut:date)
```

Exemple : BD "Gestion des intervenants" : Question avec CASE

```
SELECT pknum AS cours,
  CASE
    WHEN type='C' THEN 'Cours'
    WHEN type='TD' THEN 'Travaux dirigés'
    WHEN type='TP' THEN 'Travaux pratiques'
  END AS type_label
```

```
7 FROM tCours
```

```
1 COURS TYPE_LABEL
2 ----- ----------------
3     1 Cours
4     2 Travaux dirigés
```

1.6. Insertion de dates avec TO_DATE

Syntaxe : Gestion des dates avec TO_DATE

La gestion des dates peut poser des problèmes selon les formats paramétrés sur le serveur Oracle (en général par défaut le format est DD-MON-YY). La solution la plus rigoureuse consiste à utiliser la fonction de conversion TO_DATE.

Exemple

`TO_DATE('20021130','YYYYMMDD')` équivaut à 30-NOV-2002.

Exemple : Insertion de date dans Oracle

```
1 INSERT INTO Project (Num, Name, Begin, End)
2 VALUES (1, 'Walking on the moon', TO_DATE('20150401','YYYYMMDD'), TO_DATE('20160401',
  'YYYYMMDD');
```

Exemple : BD "Gestion des intervenants" : Insert avec date

```
1 INSERT INTO tIntervenant (pknom, prenom, poste)
2 VALUES   ('CROZAT', 'Stéphane', '4287');
3
4 INSERT INTO tCours (pkannee, pknum, titre, type, debut, fkintervenant)
5 VALUES   (2001, 1, 'Introduction','C', TO_DATE('01-01-2001','DD-MM-YYYY'), 'CROZAT');
6
7 INSERT INTO tCours (pkannee, pknum, titre, type, debut, fkintervenant)
8 VALUES   (2001, 2, 'Modélisation','TD', TO_DATE('08-01-2001','DD-MM-YYYY'), 'CROZAT');
9
10 SELECT * FROM tCours;
```

```
1 PKANNEE P TITRE          TYPE FKINTERVENANT   DEBUT
2 ------- - --------------- ---- --------------- ---------
3   2001 1 Introduction    C    CROZAT          01-JAN-01
4   2001 2 Modélisation    TD   CROZAT          08-JAN-01
```

Complément

- *TO_DATE(char)* (oracle.com)
- *TO_DATE(char)* (techonthenet.com)

1.7. Affichage à l'écran

Syntaxe

```
1 SET SERVEROUTPUT ON
```

```
1 BEGIN
2   DBMS_OUTPUT.PUT_LINE ('Hello World');
3 END;
```

1.8. Transactions en SQL

Introduction

Le langage SQL * fournit trois instructions pour gérer les transactions.

Syntaxe : Début d'une transaction

```
1 BEGIN TRANSACTION (ou BEGIN) ;
```

Cette syntaxe est optionnelle (voire inconnue de certains SGBD *), une transaction étant débutée de façon *implicite* dès qu'instruction est initiée sur la BD *.

Syntaxe : Fin correcte d'une transaction

```
1 COMMIT TRANSACTION (ou COMMIT) ;
```

Cette instruction SQL signale la fin d'une transaction couronnée de succès. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état cohérent est que les données peuvent effectivement être modifiées de façon durable.

Syntaxe : Fin incorrecte d'une transaction

```
1 ROLLBACK TRANSACTION (ou ROLLBACK) ;
```

Cette instruction SQL signale la fin d'une transaction pour laquelle quelque chose s'est mal passé. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état potentiellement incohérent et donc que les données ne doivent pas être modifiées en annulant les modifications réalisées au cours de la transaction.

Remarque : Programme

Un programme est généralement une séquence de plusieurs transactions.

2. Rappels triggers pour l'ETL

2.1. Principes des triggers

Définition : Trigger

Un *trigger* (ou déclencheur) est un bloc PL/SQL associé à une table permettant de déclencher une action avant ou après un INSERT, UPDATE ou DELETE sur cette table.

Les *triggers* sont stockés dans la base.

A quoi servent les triggers ?

- Ils permettent de renforcer l'intégrité des données (mais on préférera des contraintes "check", "unique" ou "foreign key" quand c'est possible).
- Ils permettent d'auditer des actions sur une table.
- Ils permettent de calculer des valeurs dérivées pour d'autres colonnes de la table.
  Ils constituent ainsi une des solutions pour l'implémentation des attributs dérivés.

Types de triggers

Il existe deux types de triggers :

- *Trigger sur ligne*
  le *trigger* est exécuté pour chaque ligne concernée par l'instruction insert, update ou delete (option "for each row").
- *Trigger sur instruction*
  le *trigger* est exécuté une seule fois pour l'instruction insert, update ou delete, même si elle traite plusieurs lignes d'un coup.

Syntaxe : Trigger

```
1 CREATE [OR REPLACE] TRIGGER nom_trigger {BEFORE|AFTER}
2 [INSERT OR][UPDATE [OF nom_colonne] OR][DELETE]
3 ON nom_Table
4 [FOR EACH ROW [WHEN (condition)] ]
5 DECLARE
6  [variable declarations]
7 BEGIN
8    instructions
9 END;
```

Remarque : Avant ou après ?

En général les triggers sont de type "before", en particulier pour les triggers sur ligne, c'est à dire qu'ils s'exécutent avant que l'action considérée soit exécutée, ce qui permet d'infléchir le résultat de cette action. Alors qu'un trigger "after" ne pourra plus modifier le tuple considéré et agira seulement sur d'autres tuples.

Attention : Triggers multiples

Une même table peut avoir plusieurs triggers, mais cela est à éviter en général, pour des raisons de facilité de maintenance et de performance.

Attention : Exception

Si l'exécution du trigger échoue, l'action (insert, update ou delete dans la table) est annulée (et retourne une exception Oracle).

2.2. Prédicats d'événement au sein des triggers

- INSERTING
- DELETING
- UPDATING
- UPDATING(nom_colonne)

Prédicats pour savoir dans quel contexte d'appel du trigger on est, ce qui permet dans un même trigger de s'adapter aux différents cas de déclenchement.

Rappel : BD "Gestion des intervenants" : Schéma relationnel

```
tIntervenant (#pknom:varchar, prenom:varchar, poste:integer)
tCours (#pkannee:2000..2100, #pknum:integer, titre:varchar, type:C|TD|TP, fkintervenant=>tIntervenant, debut:date)
```

Exemple : BD "Gestion des intervenants" : Trigger d'archivage de données

```
CREATE TABLE tIntervenantSav (
pknom varchar2(20) PRIMARY KEY,
prenom varchar2(20) NOT NULL
);

CREATE OR REPLACE TRIGGER trIntervenant
BEFORE DELETE OR INSERT ON tIntervenant
FOR EACH ROW
BEGIN
  IF DELETING THEN
    INSERT INTO tIntervenantSav VALUES (:old.pknom, :old.prenom);
  ELSIF INSERTING THEN
    DELETE FROM tIntervenantSav WHERE pknom = :new.pknom;
  END IF;
END;
/

DELETE FROM tCours;
DELETE FROM tIntervenant;
SELECT * FROM tIntervenantSav;
```

```
PKNOM                PRENOM
-------------------- --------------------
CROZAT               Stéphane
JOUGLET              Antoine
VINCENT              Antoine
```

```
INSERT INTO tIntervenant (pknom, prenom, poste)
VALUES   ('CROZAT', 'Stéphane', '4287');

SELECT * FROM tIntervenantSav;
```

```
PKNOM                PRENOM
-------------------- --------------------
JOUGLET              Antoine
VINCENT              Antoine
```

2.3. Manipulation des anciennes et nouvelles valeurs dans les triggers (:old et :new)

Pour les triggers de type "for each row", les colonnes de la ligne courante doivent être référencées spécifiquement selon que l'on veut l' ancienne ou la nouvelle valeur :

- `:old.nom_colonne`
- `:new.nom_colonne`

Fondamental

Il ne faut pas lire des données d'une table en cours de modification autrement que par les accès ":old" et ":new".

Attention : Anciennes valeurs en lecture seule

Il n'est jamais possible de modifier une colonne ":old".

Attention : Valeurs en lecture seule après

Pour les trigger "after", il n'est plus possible de modifier les colonnes ":new".

Remarque : Valeurs nulles

Pour les triggers "on insert" les colonnes ":old" ont la valeur NULL.

Pour les triggers "on delete" les colonnes ":new" ont la valeur NULL.

Attention

Il ne faut pas modifier de données dans les colonnes des "primary key", "foreign key", ou "unique key" d'une table.

Rappel : BD "Gestion des intervenants" : Schéma relationnel

```
1 tIntervenant (#pknom:varchar, prenom:varchar, poste:integer)
2 tCours (#pkannee:2000..2100, #pknum:integer, titre:varchar, type:C|TD|TP, fkintervenant=>tIntervenant,
  debut:date)
```

Exemple : BD "Gestion des intervenants" : Trigger de gestion de cohérence

```
1 CREATE OR REPLACE TRIGGER trCours
2 BEFORE INSERT OR UPDATE OF debut ON tCours
3 FOR EACH ROW
4 DECLARE
5   vAnneeDebut INTEGER;
6 BEGIN
7   vAnneeDebut := TO_NUMBER(TO_CHAR(:new.debut,'YYYY'));
8   IF NOT(vAnneeDebut = :new.pkannee) THEN
```

```
    :new.debut:=null;
    DBMS_OUTPUT.PUT_LINE('Inconsistency between debut and pkannee, debut set to null');
  END IF;
END;
/

SET SERVEROUTPUT ON;

INSERT INTO tCours (pkannee, pknum, titre, type, fkIntervenant, debut)
VALUES ('2001', 3, 'SQL', 'C', 'CROZAT', TO_DATE('15-01-2001','DD-MM-YYYY'));

UPDATE tCours
SET debut=TO_DATE('15-01-2002','DD-MM-YYYY')
WHERE pknum=3;

SELECT pkannee, pknum, debut FROM tCours;
```

```
TRIGGER trCours compiled
1 rows inserted.
1 rows updated.
Inconsistency between debut and pkannee, debut set to null

PKAN PKNUM DEBUT
---- ----- ---------
2001     1 01-JAN-01
2001     2 08-JAN-01
2001     3
```

2.4. Quelques règles à respecter pour les triggers

Attention

Il ne faut pas modifier de données dans les colonnes des primary key, foreign key, ou unique key d'une table.

Attention

Il ne faut pas lire des données d'une table en cours de modification autrement que par les accès :old et :new.

3. Rappels Oracle RO

3.1. Création de type en SQL3 sous Oracle (extension au LDD)

Syntaxe : Déclaration de type

```
CREATE TYPE nom_type AS OBJECT (
  nom_attribut1 type_attribut1,
  ...
);
/

```

Exemple : Création de tables d'objets (enregistrements avec OID)

```
1 CREATE TABLE t OF nom_type (
2   ...
3 )
```

Exemple : Usage des types dans les tables (modèle imbriqué)

```
1 CREATE TABLE t (
2   ...
3   nom_attribut nom_type,
4   ...
5 )
```

Complément

Héritage et réutilisation de types (cf. p.)

Méthodes de table d'objets (cf. p.51)

3.2. Création de table objet (modèles et LDD)

Définition

Une table peut être définie en référençant un type de données plutôt que par des instructions LDD * classiques. On parle alors de table objet.

Synonymes : table-objet, table d'objets

Syntaxe : Modèle logique

```
1 nom_type : <...>
2 nom_table de nom_type (#attributs_clés) autres contraintes
```

Syntaxe : LDD SQL3

```
1 CREATE TABLE nom_table OF nom_type (
2   PRIMARY KEY(attribut1),
3   attribut2 NOT NULL,
4   UNIQUE (attribut3)
5   FOREIGN KEY (attribut4) REFERENCES ...
6 );
```

Il est possible, sur une table ainsi définie, de spécifier les mêmes contraintes que pour une table créée avec une clause CREATE TABLE (contraintes de table). Ces contraintes doivent être spécifiées au moment de la création de la table, et non au moment de la création du type (bien que la définition de type permet de spécifier certaines contraintes, comme NOT NULL).

Fondamental : OID

Les enregistrements d'une table-objet peuvent être identifiés par un OID *

Complément : Méthodes

Des méthodes peuvent être associées à une table-objet.

Complément : Héritage

Cette modalité de définition de schéma permet de profiter de l'héritage de type pour permettre l'héritage de schéma de table.

Exemple

```
1 CREATE OR REPLACE TYPE typIntervenant AS OBJECT(
2 pknom varchar2(20),
3 prenom varchar2(20)
4 );
5 /
6
7 CREATE TABLE tIntervenant OF typIntervenant (
8 PRIMARY KEY(pknom),
9 prenom NOT NULL
10 );
```

3.3. Méthodes de table d'objets

Définition : Méthodes de table

Si le type sur lequel s'appuie la création de la table définit des méthodes, alors les méthodes seront associées à la table (méthodes de table).

Il sera possible d'accéder à ces méthodes de la même façon que l'on accède aux attributs (projection, sélection...).

Syntaxe : Accès aux méthodes d'une table d'objets

```
1 SELECT t.m1(), t.m2() ...
2 FROM table t
3 ...
```

Attention

L'utilisation d'un alias est obligatoire pour accéder aux méthodes.

Syntaxe : Déclaration de type avec méthodes

```
1 CREATE TYPE nom_type AS OBJECT (
2  nom_attribut1 type_attribut1
3  ...
4  MEMBER FUNCTION nom_fonction1 (parametre1 IN|OUT type_parametre1, ...) RETURN type_fonction1
5  ...
6 );
7 /
8 CREATE TYPE BODY nom_type
```

```
IS
MEMBER FUNCTION nom_fonction1 (...) RETURN type_fonction1
 IS
 BEGIN
 ...
 END ;
MEMBER FUNCTION nom_fonction2 ...
 ...
END ;
END ;
/
```

Exemple

```
CREATE TYPE typCours AS OBJECT (
 pknum NUMBER(2),
 debut DATE,
 MEMBER FUNCTION fin RETURN DATE
);
/
CREATE TYPE BODY typCours IS
MEMBER FUNCTION fin RETURN DATE
 IS
 BEGIN
   RETURN SELF.debut + 5;
 END;
END;
/

CREATE TABLE tCours OF typCours (
pknum PRIMARY KEY
);

SELECT c.pknum, c.fin()
FROM tCours c;
```

Remarque : Type retourné par une méthode

« The datatype cannot specify a length, precision, or scale. »

http://docs.oracle.com/cd/B13789_01/server.101/b10759/statements_5009.htm

3.4. Méthodes et SELF

SELF

Lorsque l'on écrit une méthode on a généralement besoin d'utiliser les attributs propres (voire d'ailleurs les autres méthodes), de l'objet particulier que l'on est en train de manipuler.

On utilise pour cela la syntaxe SELF qui permet de faire référence à l'objet en cours.

Syntaxe : SELF

```
self.nom_attribut
self.nom_méthode(...)
```

Exemple : Total d'une facture

```
1 MEMBER FUNCTION total RETURN number
2 IS
3   t number;
4 BEGIN
5   SELECT sum(f.qte) INTO t
6   FROM facture f
7   WHERE f.num=self.num;
8
9 RETURN t;
10 END total;
```

Remarque : SELF implicite

Dans certains cas simples, lorsqu'il n'y a aucune confusion possible, SELF peut être ignoré et le nom de l'attribut ou de la méthode directement utilisé.

Il est toutefois plus systématique et plus clair de mettre explicitement le self.

Exemple : Exemple de SELF implicite

```
1  MEMBER FUNCTION adresse RETURN varchar2
2  IS
3  BEGIN
4    RETURN num || rue || ville;
5  END;
```

Compléments

XII

1. Script de remise à zéro d'un schéma Oracle

```
Begin
for c in (select type_name from user_types) loop
execute immediate ('drop type ' || c.type_name || ' force');
end loop;
End;
/
Begin
for c in (select object_name from user_objects where object_type='TABLE') loop
execute immediate ('drop table ' || c.object_name || ' cascade constraints');
end loop;
End;
/
Begin
for c in (select object_name from user_objects where object_type='VIEW') loop
execute immediate ('drop view ' || c.object_name || ' cascade constraints');
end loop;
End;
/

```

2. Éléments avancés pour l'ETL

2.1. Gestion des erreurs

En cas d'erreur, il y a trois approches possibles :

1. Arrêt du chargement, traitement de l'erreur (amélioration des traitements) et reprise du chargement
   Dans un processus incrémental, c'est en général une mauvaise solution
2. Rejeter la donnée dans une table d'erreur
3. Laisser passer la donnée en prenant une décision par défaut et *logger* pour vérification

Complément

(Kimball et al., 2008, p.382-384) *

Remarque

Les approches 2 et 3 ne pose pas de problème si :

- les problèmes sont très minoritaires
- répartis sur la population des faits
- traités au fur et à mesure

Méthode : Rejeter

- Créer une copie de la structure du DW pour accueillir les données rejetées
- Ajouter un espace de stockage des commentaires (raison du rejet...)

Méthode : Laisser passer

- Adopter une approche permettant de laisser systématiquement passer les données (par exemple en joutant des valeurs d'erreur dans les dimensions)
- Logger dans une table ad hoc les cas traités par défaut

2.2. Clés artificielles

Méthode

Pour introduire des clés artificielles pour identifier les dimensions :

- Dans la zone de transformation :
  - ajouter à chaque dimension un attribut `pk`
  - ajouter une méthode `getPk()` qui renvoie `pk`
  - (supprimer la méthode qui renvoyait initialement la clé naturelle si elle n'est plus nécessaire par ailleurs)
- À chaque ajout d'un nouvel enregistrement dans une dimension
  - générer une clé artificielle dans `pk` (à l'aide d'une séquence par exemple)

- Lors de l'ajout des dimensions dans le data warehouse
    - utiliser `getPk()` pour identifier les enregistrements
- Lors de l'ajout des faits, il faut substituer les clés
    - joindre les faits avec les dimensions
    - appeler les méthodes `getPk()` de chaque dimension

Exemple

```
1 INSERT INTO dw_facts
2 SELECT d1.pk(), d2.pk()
3 FROM t_fact f, t_dim1 d1, t_dim2 d2
4 WHERE f.a=d1.a AND f.b=d1.b
```

Attention

Il faut mémoriser la correspondance entre la clé identifiant la dimension dans le système transactionnel et la clé artificielle dans le data warehouse.

Pour cela :

- soit les tables sont persistantes dans la zone T
- soit les clés du système transactionnel doivent être conservées dans le DW

Attention : Optimisation

- Les jointures à présent nécessaires diminuent les performances de chargement
- Une indexation adéquate est requise

Rappel

Les faits n'ont pas besoin d'être identifiés.

Complément : OID

Sous Oracle en RO, il est possible d'utiliser les OID à la place de clés artificielles.

Le problème sera que les OID ne sont connus qu'après insertion dans la DB, ils ne peuvent être créés dans la BDT puis transféré dans le DW.

- Il faut déclarer les tables du DW en mode RO, en conservant les clés d'origine
- Puis substituer les clés étrangères par des REF dans la table des faits, en faisant *la jointure entre la table des faits de la BDT avec les dimensions du DW*

2.3. Éléments pour l'ETL incrémental

Dans un DW incrémental le système est mis à jour régulièrement.

Une mise à jour se manifeste par :

- l'ajout de nouveaux faits

- l'éventuel ajout ou mise à jour des dimensions

Ajout de faits

Deux cas de figures :

- La source des faits est remplacée à chaque incrément (les nouveaux faits remplacent les anciens)
    - il faut ajouter les nouveaux faits au fur et à mesure en relançant l'ETL, typiquement :
        1. vider les faits de la BDT (anciens faits)
        2. repeupler les faits de la BDT (BDE->BDT)
        3. exécuter le transfert BDT->DW
- Les nouveaux faits sont ajoutés à la même source à chaque incrément :
    - il faut vider et repeupler le DW à chaque incrément
    - ou il faut une méthode qui permette de discriminer les faits déjà intégrés des nouveaux (et l'on se rapporte au cas précédent)

Méthode

Vider la BDT sera de préférence la dernière étape de l'ETL (assimilant le transfert BDT->DW à un déplacement)

1. BDE->BDT
2. BDT->DW
3. Vider BDT

Attention

Si la BDT permet de calculer des attributs d'agrégation de faits, il est nécessaire qu'elle conserve l'ensemble des données pour effectuer ses calculs.

Dans ce cas, plutôt que de vider la BDT on utilisera un attribut de discrimination (*flag*) qui mémorisera les données déjà transférées des nouvelles données.

Ajout de dimensions

L'ajout de dimensions résulte de :

- l'ajout de nouveaux faits,
- associé éventuellement à l'ajout des dimensions dans les sources transactionnelles.

De la même façon :

- soit l'on supprime et récrée la dimension
- soit l'on gère l'ajout des nouveaux enregistrements après les avoir discriminés

Mise à jour de dimensions

La mise à jour de dimension résulte de :

- l'ajout de nouveaux faits,
- associé à la mise à jour des sources transactionnelles des dimensions.

Il faut adopter une stratégie de gestion des SCD.

Méthode : Audit continu des données

Utiliser des triggers pour auditer les données et ainsi affiner les méthodes, trouver des erreurs...

2.4. Intégration des dimensions multi-sources

Méthode

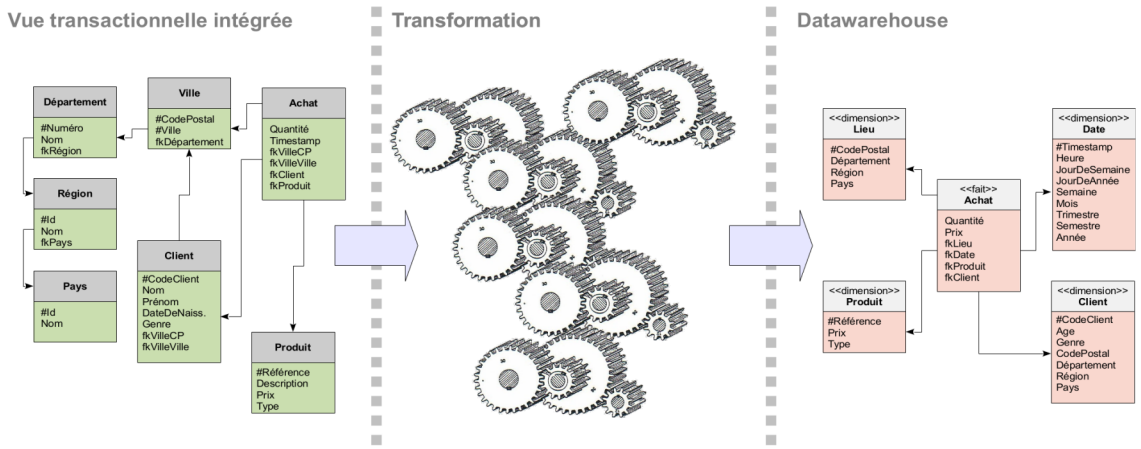
Lorsque les dimensions ont plusieurs sources, il faut :

- Contrôler les doublons (dé-duplication)
- Conformer les différentes sources

2.5. Performance et maintenance

Méthode

Il est possible de substituer cette architecture à 3 niveaux par une architecture à 2 niveaux seulement.



Architecture générale d'un ETL

3. Exercice : Gestion des erreurs [1h]

Proposer une solution pour la gestion des erreurs et des valeurs nulles de la table des faits.

Question

[solution n°1 p.60*]*

Pour chaque type d'erreur créer une nouvelle entrée dans les dimensions correspondantes.

4. Exercice : Surveillance des données

[1h]

Mettre en place des triggers afin de surveiller certaines données sensibles :

- Les dates
- Les magasins
- Typologie des erreurs
- ...

Solutions des exercices

> **Solution** n°1

Exercice p. 58

Exemple : Gestion des erreurs

Exemple d'extraits des tables du DW avec des valeurs de dimensions ajoutées pour gérées les erreurs (et l'usage de clés artificielles).

FKMag	FKPro	FKDat
12	18	32
12	41	-1
-2	55	21
...	...	...

Table des faits du DW

PK	Mag	...
-2	"Inconnu"	null
-1	"Null"	null
...	...	...
12	"M145"	...
13	"M22"	...
...	...	...

Table de la dimension Mag du DW

PK	Dat	DatStr	...
-1	null	"Tronquée"	null
...	...	...	...
21	21/1/2013	"21/01/2013"	...
...	...	...	...
32	1/2/2013	"01/02/2013"	...
...	...	...	...

Table de la dimension Dat du DW

Abréviations

API : Application Program Interface

BD : Base de Données

BDE : Base de Données d'Extraction

BDT : Base de Données de Transformation

CSV : Comma Separated Values

DM : Data Mart

DW : Data Warehouse

ETL : Extraction, Transformation, Loading

LCD : Langage de Contrôle de Données

LDD : Langage de Définition de Données

LMD : Langage de Manipulation de Données

MCD : Modèle Conceptuel de Données

MLD : Modèle Logique de Données

OID : Object Identifier

RO : Relationnel-Objet

SGBD : Système de Gestion de Bases de Données

SGBDRO : Système de Gestion de Bases de Données Relationnelles-Objets

SQL : Structured Query Language

XML : eXtensible Markup Language

Bibliographie

Kimball R., Ross M., Thornthwaite W., Mundy J., Becker B. (2008, 1998). *The Data Warehouse Lifecycle Toolkit*. Wiley Publishing, second edition.

Kimball R., Caserta J. (2004). *The Data Warehouse ETL Toolkit*. Wiley Publishing.

Webographie

Roegel Denis, *Oracle : SQL*, http://www.loria.fr/~roegel/cours/iut/oracle-sql.pdf, 1999.