

Bases de Données Avancées - Oracle

Cheikh Ba

Contenu du Cours

Objectifs

- Comprendre quelques notions avancées dans les bases de données.
- Étude de cas: le SGBD Oracle

Programme

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

Contexte

Des données ... encore des données ... toujours des données ...



RÉSERVEZ
UN BILLET DE TRAIN

Ajoutez un hôtel et faites des économies

Départ *

Paris (Toutes gares intramuros)

Arrivée *

Oriéans

Aller le *

19/03/2014

À partir de

10h

Retour

22/03/2014

À partir de

07h

EVENEMENT + TRAIN NEW !

1ère classe 2ème classe

Trains directs

Passagers

1

Passager 1

Age

26-59 ans

Carte et abonnement

Carte jeune

Programme de fidélité



SNCF: Société Nationale des Chemins de fer Français

En 2012 ...

- Chiffre d'affaires: 33,8 Milliards d'€
- 127 millions de voyageurs
- Par jour:
 - RER C: 500 000
 - RER A: 1,2 M.
 - TGV: 250 000
 - 6000 trains

👉 : Gestion informatisée **inévitable**: Achat, Abonnement, Aiguillage ...

Quelques Rappels ...

Base de Données

- Ensemble d'informations structurées.
Hiérarchiques, Relationnelles, Objet, ...
- Le marché est principalement composé de BD Relationnelles
Parfois avec une extension Objet.

SGBDR: Système de Gestion des Bases de Données Relationnelles

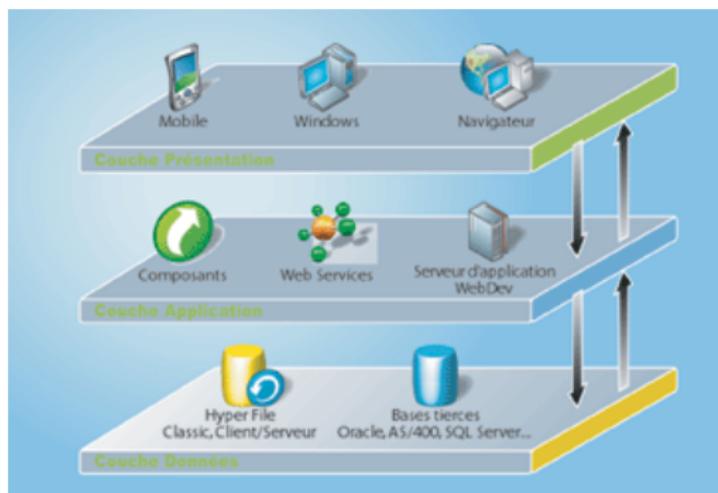
- Système (logiciel) qui permet de gérer une Base de Données
 - Type Client/Serveur: SQL Server, Oracle, MySQL, PostgreSQL, ...
 - Type fichiers partagés: Access, SQL Server CE, Paradox, DBase, ...
- Fournit les services suivants (pas tous les SGBD) :
Implémentation de SQL. Gestion des structures de données et de leur intégrité. Gestion des transactions, de la sécurité et du contrôle d'accès. Abstraction du système d'exploitation sous-jacent. Tolérance aux pannes. Administration du système. Répartition de la charge, ...

Quelques Rappels ...

Les 3 dimensions logiques d'une application

Architecture 3-tiers: extension du modèle Client/Serveur

- Couche présentation: affichage, restitution, dialogue avec l'utilisateur
- Couche métier: traitement, manipulation, gestion de la logique applicative.
- Couche accès aux données: BD, données conservées sur la durée



Oracle Corporation



- Entreprise américaine créée en 1977 par l'autodidacte L. Ellison
- Produits phares: Oracle Database (SGBD), *Oracle Weblogic Server* (serveur d'applications), *Oracle E-Business Suite* (ERP, progiciel de gestion intégré).
- Rachat en 2010 de *Sun Microsystems* à 7,4 milliards \$
 - Entreprise participant à de nombreux projets open source: java, OpenOffice.org, MySQL, ...
 - Réaction: des démissions et création de forks open source de projets.
- Quelques produits
 - Logiciels de BD: MySQL (fork *MariaDB*), Oracle Database, ...
 - Logiciels de virtualisation: Oracle VM VirtualBox
 - Solutions Java: Java EE, ME, SE, JavaFX, JDeveloper, NetBeans, ...
 - Serveurs d'applications: GlassFish, WebLogic, ...
 - Suite bureautique: OpenOffice.org (fork *LibreOffice*)



Oracle Database

- SGBDR
 - SGBDRO (SGBD Relationnel-Objet) depuis l'introduction du support du modèle objet dans sa version 8
- SGBD Leader du marché (coût d'une licence jusqu'à 200 000 €):

214 systems in ranking, February 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle ⚡	Relational DBMS	1500.23	+32.43
2.	2.	MySQL ⚡	Relational DBMS	1288.39	-8.53
3.	3.	Microsoft SQL Server ⚡	Relational DBMS	1214.27	-11.75
4.	4.	PostgreSQL ⚡	Relational DBMS	230.45	+2.20
5.	↑	MongoDB ⚡	Document store	195.17	+16.94
6.	↓	DB2 ⚡	Relational DBMS	188.46	+0.15
7.	7.	Microsoft Access ⚡	Relational DBMS	152.88	-22.11
8.	8.	SQLite ⚡	Relational DBMS	93.00	-4.29
9.	9.	Sybase ASE ⚡	Relational DBMS	87.88	-6.62
10.	10.	Cassandra ⚡	Wide column store	80.31	-0.87

- Robustesse, multi plateforme, département R&D très développé
- Jusqu'à 65536 fichiers de 128 To. Multi CPU. NoLimit (Entr. Ed.)
- Logiciels Oracle **gratuits**
 - *Oracle Database Express Edition*: taille de la base limitée à 4Go, un seul processeur utilisé.
 - *Oracle SQL Developer*: interface de gestion des objets de la base.

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

LDD - Langage de Définition de Données

Création de relations

- CREATE TABLE

```
CREATE TABLE personne (
    num_h      NUMBER
                CONSTRAINT pos_num_h CHECK (num_h > 0),
    nom        VARCHAR(30) NOT NULL,
    prenom     VARCHAR(20)
                CONSTRAINT nn_personne_prenom NOT NULL,
    parti      NUMBER,
    CONSTRAINT pk_personne_numh PRIMARY KEY(num_h),
    CONSTRAINT fk_personne_parti FOREIGN KEY(parti)
        REFERENCES parti(num_p) ON DELETE CASCADE
);
```

- **Contraintes** de domaine, d'unicité (clé secondaire), de renseignement, de clé primaire et d'intégrité référentielle

LDD - Langage de Définition de Données

Création de relations - Types des attributs 1/2

Types de données caractères

- * CHAR(n): Chaîne fixe de n octets. Max. 2000 caractères.
- * VARCHAR2(n): Chaîne variable de n octets. Max. 4000 caractères.
- * NCHAR(n): Chaîne fixe de n caractères Unicode.
- * NVARCHAR2(n): Chaîne variable de n caractères Unicode.
- * CLOB: Flot variable de caractères. Jusqu'à 4 gigaoctets.
- * NCLOB: Flot de caractères Unicode.
- * LONG: Flot variable de caractères. Jusqu'à 2 gigaoctets.
Déprécié. Fourni pour la compatibilité ascendante.

Types de données numériques

- * NUMBER[(t,d)]: Valeur numérique de t chiffres dont d décimales.
 $t + d < 38$. De $+/-1*10^{-130}$ à $+/-9.99*10^{125}$

LDD - Langage de Définition de Données

Création de relations - Types des attributs 2/2

Types de données Date/Heure

- * DATE: Date et heure du 1er jan. 4712 avant JC au 31 déc. 4712 après JC. Sur 7 octets.
- * INTERVAL YEAR (an) TO MONTH. INTERVAL DAY (jo) TO SECOND (fsec). TIMESTAMP (fsec) plus plus précis dans la définition d'un moment (fraction de seconde). TIMESTAMP (fsec) WITH TIME ZONE prend en compte les fuseaux horaires. TIMESTAMP (fsec) WITH LOCAL TIME ZONE

Types de données binaires

- * BLOB: Données binaires non structurées (images, sons, vidéo, etc.). Jusqu'à 4 gigaoctets.
- * BFILE: Données binaires stockées dans un fichier externe à la base. Jusqu'à 4 gigaoctets.
- * RAW(size): Données binaires. Jusqu'à 2000 octets. Déprécié.
- * LONG RAW : Comme RAW, jusqu'à 2 gigaoctets. Déprécié

LDD - Langage de Définition de Données

Modification de la structure d'une relation

- Ajout, suppression, modification d'attributs ou de contraintes

```
ALTER TABLE personne
```

```
    ADD (CONSTRAINT pers_nom_pre_uni UNIQUE (nom, prenom));
```

```
ALTER TABLE personne MODIFY (nom VARCHAR(50));
```

```
ALTER TABLE personne DROP CONSTRAINT pos_num_h;
```

Suppression d'une relation: DROP

- ```
DROP TABLE personne [CASCADE CONSTRAINTS];
```

  
# CASCADE CONSTRAINTS: supprime les contraintes d'intégr. réf.

## Vider une relation: TRUNCATE

- ```
TRUNCATE TABLE personne;
```


Supprime toutes les lignes en conservant la structure de la table
LDD - ne déclenche pas de trigger ON DELETE (LMD)
Même effet que DELETE FROM sans condition, mais en plus rapide

LMD - Langage de Manipulation de Données

Insertion de tuples - INSERT

- Insertion sous l'interface conversationnelle SQL

```
INSERT INTO personne VALUES (12, 'Wade', 'Abdoulaye', 6);
```

```
INSERT INTO personne (num_h, nom, prenom)  
VALUES (13, 'Sall', 'Macky');
```

```
INSERT INTO personne SELECT * FROM personnes_PDS;
```

```
INSERT INTO election (date_el)  
VALUES (to_date('2014/04/02', 'YYYY/MM/DD'));
```

- Insertion par importation de données

SQL*Loader

LMD - Langage de Manipulation de Données

Mise à jour de tuples - UPDATE

```
UPDATE personne SET parti = 6 WHERE parti = 4;
```

```
UPDATE personne SET parti = ( SELECT parti
                                FROM personne
                                WHERE nom = 'Wade' )
                                WHERE nom = 'Seck';
```

Suppression de tuples - DELETE

```
DELETE FROM personne WHERE nom = 'Wade';
```

```
DELETE FROM personne WHERE parti IN ( SELECT parti_id
                                         FROM parti
                                         WHERE sigle = 'PDS') ;
```

Objets avancés - LDD ? - LMD ?

Autres objets manipulés par Oracle

Vue (*view*) et Vue Matérialisée (Materialized View)

Synonyme (*Synonym*)

Séquence (*Sequence*)

Cluster

Index

Déclencheur (*Trigger*)

Procédure et fonction

Package

Lien (*Database Links*)

... À voir dans la partie "Objets Avancés" ...

Interfaces conversationnelles

SQL*Plus

- Utilitaire en ligne de commande: SQL, PL/SQL & scripts SQL (.sql)



SQL Developer

- Utilitaire graphique d'interrogation de BD Oracle & IDE



Oracle SQL*Plus

File Edit Search Options Help

SQL*Plus: Release 8.0.3.0.0 - Production on Wed Dec 17 15:53:55 1997

(c) Copyright 1997 Oracle Corporation. All rights reserved.

Connected to:
Oracle8 Enterprise Edition Release 8.0.3.0.0 - Production
With the Partitioning and Objects options
PL/SQL Release 8.0.3.0.0 - Production

SQL>

Oracle SQL Developer : local

File Edit View Navigate Run Debug Source Tools Help

Reports Conn...

Connections local test Help

Enter SQL Statement:

```
set sql_mode=
```

Results Script Output Explorer Autotrace

line 1: SQLPLUS Command Skipped: set sql_mode=

Script Finished Line 1 Column 13 Insert Modified Windows: CR/... Editing

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

Dictionnaire de Données

Catalogue ou Dictionnaire

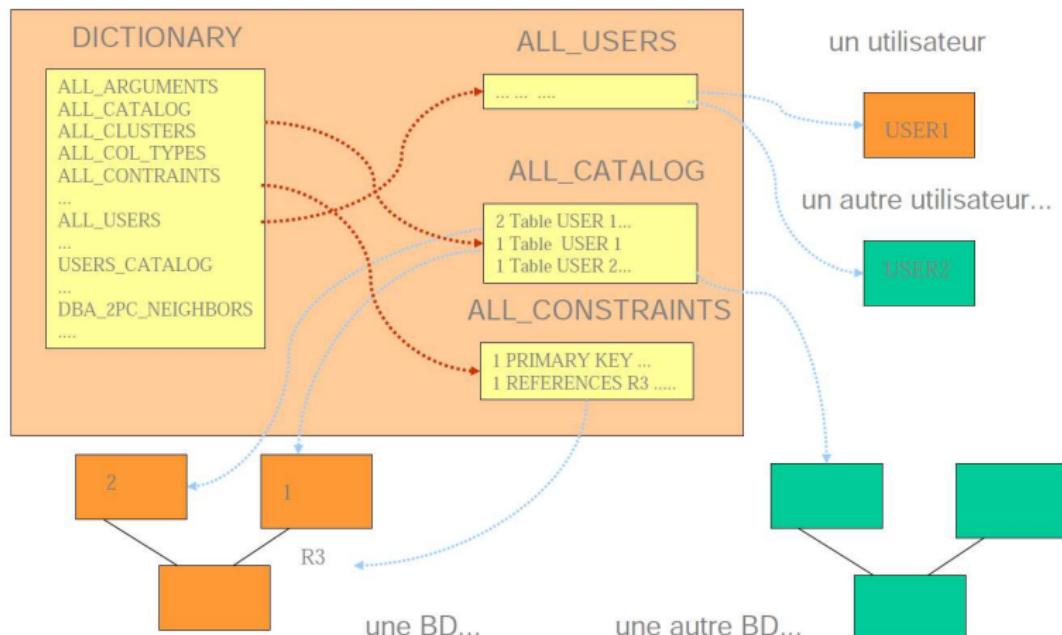
- Présent dans tout SGBD Multi-Bases et Multi-Utilisateurs :
 - Essentiel à l'administration de la base.
- Base de données **interne** gérant tous les objets connus du SGBD:
 - Les tables et leurs contraintes, les vues, les utilisateurs, les droits, les index, etc.

Structure

- Une ou plusieurs tables/vues par type d'objet géré par le SGBD
- **Une** table/vue joue souvent le rôle de point d'entrée dans cette BD d'administration : **il** liste l'ensemble des éléments du dictionnaire.
- **Exemple:**
 - Oracle: vue DICTIONARY
 - MySQL: vue INFORMATION_SCHEMA.TABLES
 - PostgreSQL: vue SYSTEM CATALOGS

Dictionnaire de Données

Le Dictionnaire (Catalogue) d'Oracle



Oracle: Dictionnaire de Données

Information cryptée accessible uniquement via des vues

USER_*	Informations sur les objets appartenant à l'utilisateur.
ALL_*	Informations sur les objets accessibles à l'utilisateur: ceux dont il est le propriétaire ainsi que ceux pour lesquels il a un droit d'accès.
DBA_*	Informations sur l'administration: droits utilisateurs, rollback segments, etc. Accessibles uniquement à l'utilisateur SYSTEM.
V\$*, GV\$*	Vues dynamiques relatives au suivi des performances.

Vue DICTIONARY

- Vue d'entrée du dictionnaire (catalogue)
- Raccourci (synonyme) : DICT

Oracle: Dictionnaire de Données

Vues principales du dictionnaire

ALL_CATALOG	Objets (tables, vues, synonymes, etc.) accessibles à l'utilisateur.
USER_CATALOG	Objets qui sont propriétés de l'utilisateur.
ALL_COL_COMMENTS	Commentaires sur le rôle des attributs des objets accessibles à l'utilisateur.
ALL_CONSTRAINTS	Contraintes d'intégrité (dont référentielles) sur les objets accessibles à l'utilisateur.
ALL_TAB_PRIVS	Droits sur les objets accessibles à l'utilisateur.
ALL_USERS	Informations sur tous les utilisateurs de la base de données.
USER_USERS	A votre avis ?

Une ou plusieurs vues par type d'objet manipulé par Oracle

- Table, vue (*view*), synonyme (*synonym*), etc.
- Vues ALL_TABLES, ALL_VIEWS, ALL_SYNONYMS, etc.

Oracle: Dictionnaire de Données

Exemple de sessions

(1) SQL> DESC dict

NAME	Null	Type
------	------	------

TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

(2) SQL> DESC all_catalog

NAME	Null	Type
------	------	------

OWNER	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLE_TYPE		VARCHAR2(11)

(3) SQL> SELECT * FROM dict WHERE table_name = 'USER_TABLES';

TABLE_NAME	COMMENTS
------------	----------

USER_TABLES	Description of the user's own relational tables
-------------	---

(4) SQL> SELECT * FROM all_tables WHERE owner = 'POLITICIEN';

OWNER	TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME
-------	------------	-----------------	--------------	-------	------

POLITICIEN	VILLE	USERS	(null)
------------	-------	-------	--------	-------	------

Oracle: Dictionnaire de Données

Documentation BD Oracle: commentaires stockés

ALL_TAB_COMMENTS: Commentaires sur les tables/vues accessibles

ALL_COL_COMMENTS: Commentaires sur les attributs des tables/vues

Commenter un objet créé (SQL Oracle)

- Commentaire sur une table

```
COMMENT ON TABLE <nom_table>
    IS 'texte de commentaire' ;
```

- Commentaire sur un attribut

```
COMMENT ON COLUMN <nom_table>. <nom_colonne>
    IS 'texte de commentaire' ;
```

Oracle: Pseudo Table - Pseudo Colonne

Pseudo Table: DUAL

- Table spéciale, en lecture seule, avec une ligne et une colonne, présente par défaut dans toutes les BD Oracle. Couramment sollicitée, mais les interrogations ne portent jamais sur sa colonne.

```
SQL> DESC dual
Name      Null    Type
-----  -----
DUMMY          VARCHAR2(1)
```

```
SQL> SELECT user FROM dual;
USER
-----
ETUDIANT
```

```
SQL> SELECT to_char(sysdate, 'DD MONTH YYYY, HH24:MI:SS') AS zz FROM dual;
ZZ
-----
02, APRIL 2014, 17:20:58
```

```
SQL> SELECT * FROM dual;
DUMMY
-----
X
```

```
SQL> SELECT sysdate FROM dual;
SYSDATE
-----
02-APR-14
```

Oracle: Pseudo Table - Pseudo Colonne

Pseudo Colonnes

- Pas de vraies colonnes de table, mais se comportent comme telles.
- SYSDATE, SYSTIMESTAMP, UID, USER, CURRVAL, NEXTVAL, LEVEL, ROWID, ROWNUM, ORA_ROWSCN

```
SQL> SELECT systimestamp, uid, user FROM dual;
```

SYSTIMESTAMP	UID	USER
02-APR-14 06.01.18.706199000 PM +00:00	37	ETUDIANT

```
SQL> SELECT rownum, rowid, num_v, nom FROM ville WHERE rownum <= 3;
```

ROWNUM	ROWID	NUM_V	NOM
1	AAADaWAAEAAAAIsAAD	1	Dakar
2	AAADaWAAEAAAAIsAAE	2	Saint-Louis
3	AAADaWAAEAAAAIsAAF	45	Diourbel

ROWID: adresse physique de la ligne: adr. bloc + rang dans le bloc, ...

sequence_name.CURRVAL -- Donne le dernier numéro de séquence utilisé.

sequence_name.NEXTVAL -- Donne le prochain numéro de séquence.

Importation de données

Importation à partir de fichiers de données

- Données sous la forme d'un ensemble de fichiers ASCII/Unicode
- Données non structurées. Données suivant un certain format.
 - Exemple du format CSV: suite de champs de valeurs séparés par ";"

PDS;Parti Démocratique Sénégalais;droite

PS;Parti Socialiste;gauche

...

- BD à charger ayant sa propre structure interne

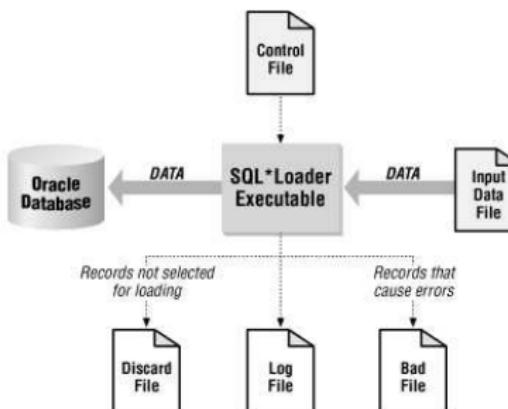
Utilitaire:

- Lecture structurée des données et chargement cohérent
 - Valeurs chargées dans les bons attributs de la base, conversion de type
 - Ajout de valeur de clé primaire, etc.
 - Exemple: **SQL*Loader** du SGBD Oracle

Importation de données

Principe de SQL*Loader

- Fichier de contrôle: formats des données à lire et de la BD à charger

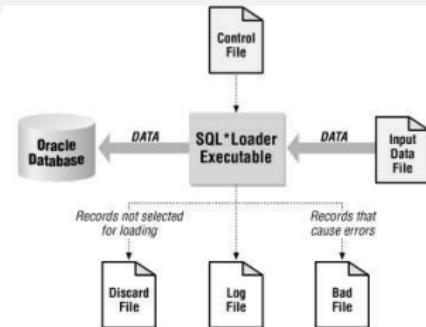
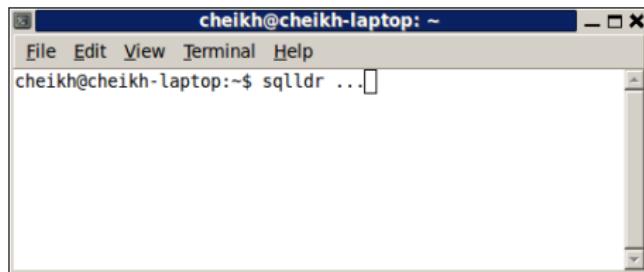


- Log File:** enregistre les activités SQL*Loader durant un *Loading*
 - Champs chargés. Nbre de lignes lues, rejetées. *Loading time*, etc.
- Bad File:** enregistre les erreurs SQL Loader
 - Violations de contraintes d'intégrité. Tablespace Full, etc.
- Discard File:** lignes ne répondant pas aux critères de chargement.

Importation de données

- 0;PS;Parti Socialiste;gauche;0
0;PC;Parti Communiste;gauche;0
0;Verts;Parti Écologiste;gauche;0
0;Modem;Mouvement Démocrate;centre;0
0;UMP;Union Majorité Présidentielle;droite;1
0;FN;Front National;droite;0
#####
 # Fichier De #
Données #
#####
- Relation: PARTI (id, sigle, nom, courant, gvt)
Clé: id.
Type de données: id, gvt : Entier.
sigle, nom, courant : Chaîne Car.
Contrainte : gvt dans {0, 1}.
#####
 # Base De Données #
#####
- LOAD DATA
INFILE 'parti.don'
INSERT INTO TABLE parti
FIELDS TERMINATED BY ";"
(id "ma_sequence_pour_id.nextval", -- (*)
sigle CHAR "upper(:sigle)" ,
nom CHAR ,
courant CHAR,
gvt INTEGER EXTERNAL) -- Commentaires: ...
● (*) Attention: 1er champ ignoré, contrairement au délimiteur par POSITION

Importation de données - SQL*Loader



Appel

- Commande sqlldr
 - Spécification des paramètres par position


```
sqlldr userid/password fic_ctl.ctl [fic_data] [fic_log] [fic_bad]
```

```
sqlldr userid/password fic_ctl.ctl      # ==> génère les autres noms
```
 - Spécification des paramètres par mots-clés


```
sqlldr keyword=value [, keyword=value, ...]
```

```
sqlldr control=fic.ctl, userid=etudiant/ubuntu, bad=fic.bad
```
 - Aide en ligne


```
sqlldr
```

Importation de données - SQL*Loader

```
$ sqlldr
```

```
SQL*Loader: Release 10.2.0.1.0 - Production on Sat Mar 29 10:21:43 2014
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Usage: SQLLDR keyword=value [, keyword=value, ...]
```

Valid Keywords:

```
userid -- Oracle username/password
control -- Control file name
    log -- Log file name
    bad -- Bad file name
    data -- Data file name
discard -- Discard file name
discardmax -- Number of discards to allow          (Default all)
    skip -- Number of logical records to skip      (Default 0)
    load -- Number of logical records to load      (Default all)
errors -- Number of errors to allow              (Default 50)
rows -- Number of rows in conventional path bind array or between
      direct path data saves
      (Default: Conventional Path 64, Direct path all)
....
```

Importation de données - SQL*Loader

Localisation des données

- Données intégrées au fichier de contrôle



```
LOAD DATA
INSERT INTO TABLE parti
FIELDS TERMINATED BY ";"
( num_p      INTEGER EXTERNAL,
  ...
)
BEGIN DATA
1;PT;Parti des Travailleurs;ext_gauche;0
2;LO;Lutte Ouvrière;ext_gauche;0
...
```

- Localisation du fichier de données spécifiée dans le fichier de contrôle

```
LOAD DATA
INFILE 'parti.don'
INSERT INTO TABLE parti
FIELDS TERMINATED BY ";"
( num_p      INTEGER EXTERNAL,
  ...
)
```

- Fichier de données spécifié dans la ligne de commande
 - Réutilisation du fichier de contrôle



`sqlldr ... , data=parti.don, ...`

Importation de données - SQL*Loader

Segmentation du fichier de données (attributs/champs)

1/4

- À l'aide de **séparateurs**

```
# FIELDS TERMINATED BY {WHITESPACE | X'hex_digits' | 'string'}
```

- Par leurs **positions** au sein de chaque ligne.

1 PT	Parti des Travailleurs	ext_gauche
2 LO	Lutte Ouvrière	ext_gauche
3 LCR	Ligue Communiste Révolutionnaire	ext_gauche
14 CPNT	Chasse Pêche Nature Traditions	droite

Fichier de Contrôle

```
LOAD DATA
INFILE 'parti.don'
INSERT INTO TABLE parti
(
    num_p      POSITION(1:2) INTEGER EXTERNAL,
    sigle     POSITION(04:07) CHAR,
    nom       POSITION(15:46) CHAR,
    courant   POSITION(48:57) CHAR
)
```

Importation de données - SQL*Loader

Segmentation du fichier de données (attributs/champs)

2/4

- Délimiteur spécifique à un champ - délimiteur pour tous les champs

# Données #	# Fichier de Contrôle #
Paris,462440N 0863825W	...
Gambie,472406N 0874242W	FIELDS TERMINATED BY ',' (county CHAR, latitude CHAR TERMINATED BY ' ', longitude CHAR
)

- Enclosing Characters*

[OPTIONALLY] ENCLOSED BY {'string' | X'hex_digits'} [AND idem]

Données
"Baraga, Township of",civil,Baraga,464419N,0883419W,0,200012081404
"Ontonagon, Township of",civil,Ontonagon,464904N,0891640W,0,2000120
Anna River,stream,Alger,462440N,0863825W,630,200012081058

Fichier de Contrôle
... FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ...

Importation de données - SQL*Loader

Segmentation du fichier de données (attributs/champs)

3/4

- Traitement des *Nulls*

... FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '''
 (feature_name, feature_type, county, latitude, longitude, elevation)
 #####

- (1) "Escanaba Light","locale","Delta","454448N","0870213W","582"
- (2) "Vulcan Lookout Tower","tower","Dickinson",,,0
- (3) Driggs River,stream,Schoolcraft,461134N,0855916W,""
- (4) Seney Nat. Wildlife Refuge,park,Schoolcraft,461600N,0860600W,

Ligne 2: latitude = longitude = NULL

Ligne 3: elevation = NULL

Ligne 4: erreur:

Record 4: Rejected - Error on table
 MICHIGAN_FEATURES, column ELEVATION.
 Column not found before end of logical
 record (use TRAILING NULLCOLS)

- TRAILING NULLCOLS:** champs manquants (fin d'enregistrement) = **NULL**

Importation de données - SQL*Loader

Segmentation du fichier de données (attributs/champs)

4/4

- Ignorer des champs:

- Champs que l'on ne veut pas charger
- Clause **FILLER**^a

```

fields: state code, feature name, feature type, county name
        ###
        "MI","Maxton Plains","flat","Chippewa"
        "MI","Marble Head","summit","Chippewa"
        "MI","Potagannissing River","stream","Chippewa"
        ###
        ...
FIELDS TERMINATED BY ',' ENCLOSED BY ''
(
    champ_1 FILLER,          -- ne sera pas chargé
    feature_name,
    champ_2 FILLER,          -- ne sera pas chargé
    county
)

```

^aÀ partir de la version Oracle8i

Importation de données - SQL*Loader

Chargement conditionnel des données - Chargement multitable

```
LOAD DATA
INFILE 'parti.don'
INSERT
INTO TABLE pers_fr
    WHEN pays='fr'
    (
        pays      POSITION(01:02)
        nom       POSITION(05:25)
    )
INTO TABLE pers_etranger
    WHEN pays !='fr'
    (
        pays      POSITION(01:02)
        nom       POSITION(05:25)
    )
```

Importation de données - SQL*Loader

Modification des données avant chargement

- Les fonctions SQL standard sont implantées dans SQL*Loader

```

LOAD DATA
INFILE 'parti.don'
INSERT
INTO TABLE parti
FIELDS TERMINATED BY ";"
TRAILING NULLCOLS
(
    name          CHAR "upper(:name)" ,
    birthday      DATE "DD-MM-YYYY" ,
    fortune_fr   INTEGER EXTERNAL,
    fortune_euro  INTEGER EXTERNAL ":fortune_fr / 6.56" ,
    pers_id       "ma_sequence_pour_id.nextval"           -- (*)
)

```

- (*) Attribut en fin de liste d'attributs, avec TRAILING NULLCOLS.
Si attribut en début, le 1er champ du fichier de données est ignoré.
SAUF si on utilise un délimiteur par POSITION.

Importation de données - SQL*Loader

Principaux formats externes reconnus par SQL*Loader

Nom	Descriptif	Type SQL Oracle
Types portables - données sous forme de chaîne de caractères		
CHAR	Chaîne de caractères	VARCHAR2, CHAR,
DATE	Date sous forme de chaîne de caractères. Parfois fournie avec un masque de représentation	DATE
INTEGER EXTERNAL	Entier sous forme de caractères	Types numériques
FLOAT EXTERNAL	Réel sous forme de caractères	Types numériques
DECIMAL EXTERNAL	Décimal sous forme de caractères	Types numériques
Types non portables - dépendent du hardware: taille, codage du signe, etc.		
INTEGER	Entier sous forme binnaire	Types définis dans
FLOAT	Réel sous forme binnaire	les langages de
DECIMAL	Décimal sous forme binnaire	programmation

Importation de données - SQL*Loader

Différents modes de chargement

```
LOAD DATA
INFILE 'parti.don'
INSERT INTO TABLE parti -- ou "APPEND" - "REPLACE" - "TRUNCATE"
(
    num_p      POSITION(1:2) INTEGER EXTERNAL,
    ...
)
```

- **INSERT**

Requires that the table being loaded be empty.

- **APPEND**

Allows you to load data into an existing table regardless of whether the table contains data or is empty.

- **REPLACE**

Deletes existing data from the table being loaded. DELETE statement.

- **TRUNCATE**

Truncates the table being loaded. SQL TRUNCATE statement.

Oracle Import - Oracle Export

Présentation

- Outil de transfert du contenu d'une base vers une autre
- Sauvegarde partielle du contenu d'une base (format binaire propriét.)
- Ne concerne que des bases Oracle.

Lancement

- Commande:

```
exp nom_user/password
```

Il y a d'autres versions et déclinaisons: Pump Export, ...

- Outils (Exemple *Oracle SQL Developer*)
 - Génération de fichiers textes (.sql) de LDD et/ou LMD

Oracle Import - Oracle Export

Coût de traitement

- Opération lourde en terme de calculs
- Cette opération peut encore être ralentie lors du chargement
 - Vérification des contraintes
 - Remise à jour des index
 - Vérification des conditions de déclenchement des triggers

Recommandations

- Désactiver les contraintes avant le chargement et les réactiver ensuite
- Idem pour les index et les triggers

LDD

ALTER TABLE ...

Oracle Import - Oracle Export

Désactivation de contraintes

Désactivation

```
ALTER TABLE <table> DISABLE CONSTRAINT <nom_contrainte>
```



Importation

```
sqlldr
```



Préparation ré-activation

Création d'une table de récupération des tuples posant problème



Réactivation

```
ALTER TABLE <table> ENABLE CONSTRAINT <nom_contrainte>
EXCEPTIONS INTO <table_erreurs>
```

Oracle: Désactivation de contraintes

Cas d'étude: table ADULTES

1/3

```
CREATE TABLE adultes (
    num_h      NUMBER PRIMARY KEY,
    nom        VARCHAR2(30),
    age        NUMBER,
    CONSTRAINT nn_nom     CHECK (nom IS NOT NULL),
    CONSTRAINT age_adulte CHECK (age >= 18)
);
```

```
SQL> SELECT * FROM user_constraints WHERE table_name = 'ADULTES';
```

OWNER	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
POLITICIEN	NN_NOM	C	nom is not null
POLITICIEN	AGE_ADULTE	C	age >= 18
POLITICIEN	SYS_C004088	P	(null)

Oracle: Désactivation de contraintes

Cas d'étude: table ADULTES

2/3

```
SQL> SELECT * FROM adultes;
```

NUM_H	NOM	AGE
1	DIOP	90
2	FALL	19

```
SQL> INSERT INTO adultes  
VALUES (3, 'SARR', 5);
```

```
SQL Error: ORA-02290 check constraint  
(POLITICIEN.AGE_ADULTE) violated
```

```
SQL> ALTER TABLE adultes DISABLE CONSTRAINT age_adulte;  
-- table ADULTES altered.
```

```
SQL> SELECT status FROM user_constraints  
WHERE constraint_name = 'AGE_ADULTE';  
  
STATUS  
-----  
DISABLED
```

```
SQL> INSERT INTO adultes VALUES (3, 'SARR', 5); -- 1 row inserted  
SQL> INSERT INTO adultes VALUES (4, 'NDIAYE', 10); -- 1 row inserted
```

Oracle: Désactivation de contraintes

Cas d'étude: table ADULTES

3/3

```
SQL> CREATE tab_problemes (
      adresse          ROWID,
      utilisateur     VARCHAR2(30),
      nom_table       VARCHAR2(30),
      nom_constrainte VARCHAR2(30)
);
                                         #####
```

-- table TAB_PROBLEMES created

```
SQL> ALTER TABLE adultes ENABLE CONSTRAINT age_adulte
      EXCEPTIONS INTO tab_problemes;
```

SQL Error: ORA-02293: cannot validate (POLITICIEN.AGE_ADULTE) ...

```
SQL> SELECT * FROM tab_problemes;
```

ADRESSE	UTILISATEUR	NOM_TABLE	NOM CONTRAINTE
AAADXuAAEAAAAIPAAC	POLITICIEN	ADULTES	AGE_ADULTE
AAADXuAAEAAAAIPAAD	POLITICIEN	ADULTES	AGE_ADULTE

Oracle: Désactivation de contraintes

Désactivation temporaire d'une contrainte

- Mise en état DISABLE de la contrainte

```
ALTER TABLE <nom_table> DISABLE CONSTRAINT <nom_contrainte>
```

- Statut observable dans le dictionnaire

```
ALL_CONSTRAINTS, USER_CONSTRAINTS.
```

Réactivation de la contrainte

- Mise en état ENABLE de la contrainte

```
ALTER TABLE <nom_table> ENABLE CONSTRAINT <nom_contrainte>
[EXCEPTIONS INTO <nom_table_erreur>];
```

- Liste des données violant la contrainte dans une table d'erreurs

Oracle: Désactivation de contraintes

Table des erreurs récupérées

- Table à créer spécifiquement à l'aide de la commande CREATE TABLE
- Respect d'une structure particulière
 - 1^{er} champ de type ROWID: contiendra les adresses des tuples erronés
 - 2^{ème} champ de type VARCHAR2(30): nom du propriétaire de la table
 - 3^{ème} champ de type VARCHAR2(30): nom de la table
 - 4^{ème} champ de type VARCHAR2(30): nom de la contrainte violée

Type ROWID: rappel

- Type prédéfini spécifique à Oracle et correspondant à un pseudo-attribut qui précise l'adresse de chaque tuple.
- Qu'une relation ait ou pas une clé primaire, Oracle lui associe un attribut ROWID unique (*Row Identifier*).

Oracle: Désactivation de contraintes

Consultation des erreurs à traiter pour une bonne réactivation

- Jointure entre les deux tables pour récupérer toutes les informations

```
SQL> SELECT * FROM adultes
      WHERE rowid IN (SELECT adresse FROM tab_problemes);
```

NUM_H	NOM	AGE
3	SARR	5
4	NDIAYE	10

Modes **VALIDATE / NOVALIDATE**

- Vérification ou non pour les tuples déjà présents dans la table

```
SQL> ALTER TABLE adultes ENABLE NOVALIDATE CONSTRAINT age_adulte;
-- table ADULTES altered.
```

Pas de vérification de la contrainte sur les données anciennes !

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

Objets avancés

Dès que l'on quitte le domaine des SGBD mono-utilisateurs, un certain nombre d'objets n'ayant pas de lien direct avec le modèle relationnel est nécessaire pour une utilisation et une administration efficace

- Vue
- Vue matérialisée
- Index
- Déclencheur (*trigger*)

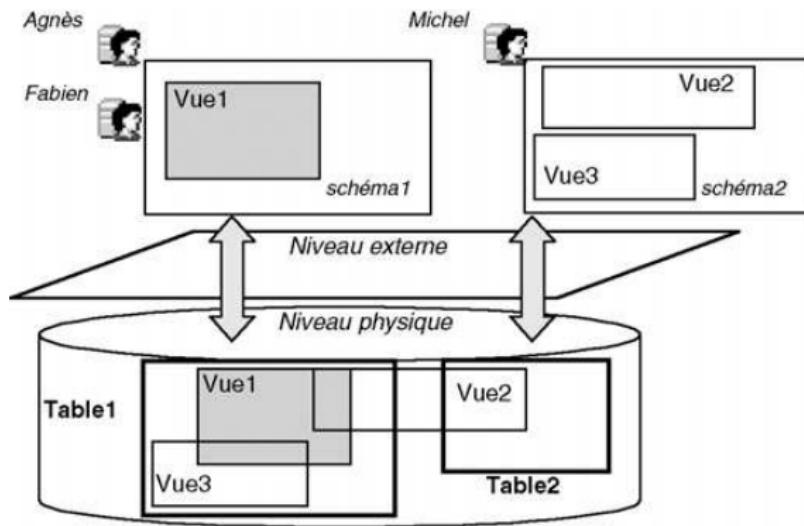
Objets spécifiques (??) à Oracle

- Synonyme
- Séquence
- Cluster
- ...

Vues

Principe

- Fenêtre sur le contenu de la base de données.
- Relation virtuelle: pas de duplication de données
- Mémorisation de l'expression définissant la vue, et non de son contenu



Vues

Exemple

```
CREATE VIEW VilleFr AS
    SELECT Nom FROM Ville WHERE Etat='France';
```

The diagram illustrates the creation of a view. On the left, there is a table with four columns: No, Nom, Etat, and Population. The data consists of four rows: (1, Lyon, France, 1,2), (2, Paris, France, 10), (3, Mexico, Mexique, 18), and (4, Paris, Texas, 0,05). An arrow points from this table to a smaller table on the right, which only contains the Nom column with the values Lyon and Paris.

No	Nom	Etat	Population
1	Lyon	France	1,2
2	Paris	France	10
3	Mexico	Mexique	18
4	Paris	Texas	0,05

Nom
Lyon
Paris

- Voir les données de différentes manières
 - Exemple: calcul de données de synthèse avec fonctions d'agrégat


```
CREATE VIEW Pop_Pays AS
              SELECT Etat, Avg(Pop) FROM Ville GROUP BY Etat;
```
- Sécurité vis à vis des données cachées
 - Exemple: n'autoriser que l'accès aux vues, pas aux tables

Vues

Abréviaction de requête

- Décomposition d'une requête en sous-requêtes (résultat intermédiaire)
- Vue intermédiaire: peut avoir une influence bénéfique sur le temps d'exécution d'une requête (sauvegarde du plan d'exécution)

VILLE	No	Nom	Etat	Pop
	1	Lyon	France	1,2
	2	Paris	France	10
	3	Mexico	Mexique	18
	4	Paris	Texas	0,05

PAYS	Etat	Pop
	France	62
	Turquie	90
	Mexique	120
	Texas	30

- Exemple: rechercher les villes de plus d' 1 million d'habitants dans les pays de plus de 50 millions d'habitants
 - Requête avec jointure
 - Requête imbriquée
 - Requête avec vue intermédiaire: rôle de macro ou fonction

Vues - Manipulation SQL - Conformité avec SQL3

Création

- Définition comme résultat d'une requête de sélection SQL
- Syntaxe:

```
CREATE VIEW <nom_vue> AS <requête SQL>
```

```
[ WITH { CHECK OPTION | READ ONLY } ] ;
```

Contraintes sur les colonnes: Oui, mais Oracle n'assure pas encore l'activation. Elles sont créées avec l'option DISABLE NOVALIDATE

- Vérification des conditions sur mise à jour (CHECK OPTION)
 - Par défaut, pas de contrôle des conditions de définition de la vue
 - Option de vérification fortement recommandée

Suppression

- Syntaxe:

```
DROP VIEW <nom_vue> ;
```

Catalogue: USER_VIEWS, ALL_VIEWS

Vues - Utilisation

Consultation (SELECT)

- Aucun problème

Mise à jour d'une table via une vue (UPDATE, INSERT, DELETE)

1/2

- Mise à jour avec vérification des contraintes d'intégrité
- Problèmes éventuels de complétion des données.

```
CREATE VIEW Ville_Fr AS
```

```
    SELECT Nom FROM Ville WHERE Etat='France' WITH CHECK OPTION;
```

No	Nom	Etat	Population
1	Lyon	France	1,2
2	Paris	France	10
3	Mexico	Mexique	18
4	Paris	Texas	0,05

Vues - Utilisation

Mise à jour d'une table via une vue (UPDATE, INSERT, DELETE)

2/2

```
SQL> CREATE VIEW Ville_Fr AS
      SELECT Nom FROM Ville WHERE Etat='France' WITH CHECK OPTION;
SQL> INSERT INTO Ville_Fr VALUES ('Blois');
SQL Error: ORA-01400: cannot insert NULL into "VILLE"."NO" ...
```

No	Nom	Etat	Population
1	Lyon	France	1,2
2	Paris	France	10
3	Mexico	Mexique	18
4	Paris	Texas	0,05

- SQL3:

- Vue multitable: mise à jour impossible
- Vue monutable: possible si attribut clé (+ champs avec NOT NULL) présents dans la vue + absence de clause DISTINCT ou fonction d'agrégat (AVG, COUNT, MAX, etc.) + pas de GROUP BY, ORDER BY, HAVING, etc.

Vues matérialisées

Principe

Vue "réelle": fenêtre sur le contenu de la base de données MAIS ...

- Objet réel: recopie physique des données concernées
- Actualisation périodique: rafraîchissement avec les données de la BD

	Vue	Vue concrète
Données observées	virtuelles	réelles
Accès aux données sources	direct	indirect
Mise à jour des données	immédiate	périodique / sur demande

Utilisation

- Pré-agrégation des données - data warehouse (entrepôt de données)
- Sécurisation des données - plus d'accès aux données sources

Vues matérialisées

Catalogue

- SNAPSHOT - CLICHE: avant la version 10g.
- MATERIALIZED VIEW:
 - USER_MVIEWS, ALL_MVIEWS

Paramétrisation des vues concrètes

- Chargement des données à la création ou lors de la première requête
- Fréquence de rafraîchissement des données
- Mode de rafraîchissement des données: recopie complète ou incrémentale (plus rapide).
- Optimisation par réécriture de requête: permettre à l'optimiseur de requête de réécrire son plan d'exécution pour accélérer l'exécution d'une requête "proche" de celle de la vue

Vues matérialisées

Création (syntaxe simplifiée)

1/2

```
CREATE MATERIALIZED VIEW <nom_vue_concrete>
  BUILD [IMMEDIATE | DEFERRED]
  [ENABLE QUERY REWRITE]
  REFRESH [FAST | COMPLETE | FORCE | NEVER]
  ([ON COMMIT | ON DEMAND]) | ([START WITH date] [NEXT date]])
  AS <requête SQL> ;
```

- **BUILD**
 - IMMEDIATE: chargement immédiat.
 - DEFERRED : chargement à la prochaine opération REFRESH
SQL> execute DBMS_MVIEW.REFRESH ('<nom_vue_concrete>')
- **ENABLE QUERY REWRITE:** amélioration de l'interprétation des ordres SQL semblables à celui de la vue.

Vues matérialisées

Création (syntaxe simplifiée)

2/2

- Modes de rafraîchissement
 - FAST: mise à jour incrémentale
 - COMPLETE: mise à jour intégrale
 - FORCE: FAST pas défaut et COMPLETE si impossible
 - NEVER: pas de rafraîchissement.
- Modes FAST et FORCE: obligation de créer des journaux qui "suivent" les modifications effectuées sur toutes les tables sources

```
CREATE MATERIALIZED VIEW LOG ON table1 WITH ...
    INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW LOG ON table2 ...
```

- Dictionnaire de données: ALL_MVIEWS_LOG

Suppression

```
DROP MATERIALIZED VIEW <nom_vue_concrete> ;
```

Vues matérialisées

Exemple de sessions

```
SQL> CREATE MATERIALIZED VIEW mavueM  
REFRESH FORCE  
START WITH SYSDATE NEXT (SYSDATE + 1/2)      # Chaque 12 heures  
AS SELECT nom, traduction  
        FROM ville, refville  
       WHERE ville.num_v = refville.ville;
```

```
# materialized view MAVUEM created
```

```
SQL> SELECT query FROM USER_MVIEWS;
```

QUERY

```
select nom, traduction from ville, refville where ville.num_v = ...
```

Synonymes

Utilisation

- Alias d'un objet Oracle: table, vue, séquence, procédure, etc.
- Confidentialité
- Raccourci pour simplifier l'accès aux objets
- Maintenabilité: la nature du synonyme peut être modifiée sans mettre à jour tous les programmes qui l'utilisent.

Création

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM <nom_synonym> FOR <nom_objet>;
```

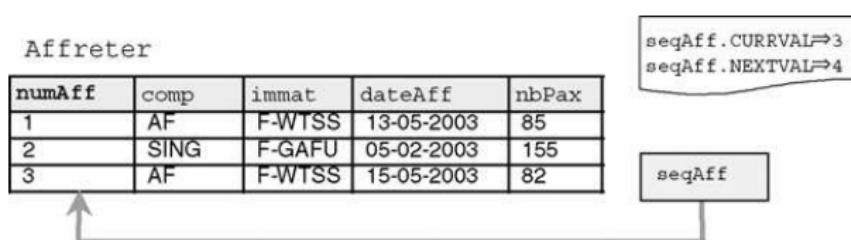
Suppression

```
DROP SYNONYM <nom_synonym>;
```

Séquence

Présentation

- Objet virtuel qui ne contient aucune donnée
- Utilisé pour générer automatiquement des valeurs (NUMBER).
 - Clés primaires par exemple.
- Gérée indépendamment des tables, mais utilisable pour plusieurs tables et par plusieurs utilisateurs.



- Deux fonctions (ou pseudo-colonnes ou directives)
 - CURRVAL: retourne la valeur courante (après un 1er appel de NEXTVAL)
 - NEXTVAL: incrémente la séquence et retourne la valeur obtenue

Séquence - Création

Syntaxe

```
CREATE SEQUENCE nomSéquence  
[INCREMENT BY entier ]  
[START WITH entier ]  
[ { MAXVALUE entier | NOMAXVALUE } ]  
[ { MINVALUE entier | Nominvalue } ]  
[ { CYCLE | NOCYCLE } ]  
[ { CACHE entier | NOCACHE } ]  
[ { ORDER | NOORDER } ] ;
```

INCREMENT BY: intervalle entre deux val. Non nul. 1 par défaut.

START WITH: 1ère valeur. MINVALUE ou MAXVALUE par défaut (Séq asc ou desc)

MAXVALUE: <= 10^29 - 1. MINVALUE: >= -10^27 - 1

NOMAXVALUE: par défaut. Fixe le max à 10^29 - 1 (asc) ou à -1 (desc)

Nominvalue: par défaut. Fixe le min à 1 (asc) ou à -10^27 - 1 (desc)

CYCLE: génération continue de valeurs même après les limites.

NOCYCLE: par défaut. Ne doit plus générer de valeur après les limites

CACHE: pré-allocation de mémoire (valeurs). NOCACHE. 20 si abs des deux.

ORDER: Garantie val. générées dans l'ordre des req. NOORDER (par défaut)

Séquence

Utilisation

```
SQL> CREATE SEQUENCE maSequence
      INCREMENT BY    1
      START WITH     1
      MINVALUE       0
      MAXVALUE       3
      CYCLE
      NOCACHE;
# sequence MASEQUENCE created
```

```
SQL> SELECT masequence.nextval FROM dual
NEXTVAL
-----

```

```
1
```

```
SQL> SELECT masequence.nextval FROM dual
NEXTVAL
-----

```

```
2
```

```
SQL> INSERT INTO ville VALUES (masequence.currrval, 'Kedougou', 500000);
# 1 rows inserted
```

Séquence

Modification

```
SQL> ALTER SEQUENCE nomSéquence  
[INCREMENT BY entier ]  
[ { MAXVALUE entier | NOMAXVALUE } ]  
[ { MINVALUE entier | NOMINVALUE } ]  
[ { CYCLE | NOCYCLE } ]  
[ { CACHE entier | NOCACHE } ]  
[ { ORDER | NOORDER } ] ;
```

```
SQL> ALTER SEQUENCE masequence INCREMENT BY 5 MAXVALUE 650;  
# sequence MASEQUENCE altered.
```

Suppression

```
DROP SEQUENCE nomSéquence ;
```

Catalogue

```
USER_SEQUENCES , ALLSEQUENCES .
```

Triggers - Déclencheurs

Trigger

- Action se déclenchant automatiquement suite à un événement ou à un changement d'état de la BD
- Action: ordre SQL ou procédure stockée. Sur des tables ou des vues
- 3 parties : (1) Description de l'événement traqué, (2) Conditions de déclenchement (3) Action à réaliser.

Différents types de déclencheurs

- Sur ordre LMD: INSERT, UPDATE, DELETE.
- Sur ordre LDD: CREATE, ALTER, DROP, GRANT, COMMENT, ...
- Sur événement SGBD: démarrage, arrêt, ...

Applications

- Règles de gestion non modélisables par des contraintes d'intégrité
- Chargement automatique de tables d'audit sur l'évolution de la BD

Triggers - Déclencheurs

Création: Syntaxe simplifiée

```
CREATE [ OR REPLACE ] TRIGGER nom_trigger
BEFORE | AFTER | INSTEAD OF      -- INSTEAD OF ==> Vue multitable
INSERT | DELETE | UPDATE [ OF liste_colonne ]
ON {[schéma.]nom_table | nom_vue}
[FOR EACH ROW]
[WHEN condition]
BEGIN
    code_trigger                  -- Bloc PL/SQL
END;
```

Activation / Désactivation

```
ALTER TRIGGER nom_trigger ENABLE | DISABLE
```

Suppression

```
DROP TRIGGER nom_trigger
```

- Catalogue: ALL_TRIGGERS, USER_TRIGGERS
- Étude approfondie: Partie PL/SQL (langage propre à ORACLE)

Index

Optimisation

- BD réelle: très grosse masse de données impliquant une exécution des requêtes coûteuses en temps.
- Index: une solution d'optimisation en accélérant l'accès aux données les plus souvent utilisées

Index

- **Principe:** structure de données associant directement à une valeur de clé donnée l'adresse physique du tuple considéré dans la BD
- **Organisation d'index:** brut, trié, hiérarchisé, avec hachage
- Index sur clé primaire mais également sur champs très utilisés comme les clés secondaires par exemple.

Catalogue

- ALL_INDEXES, USER_INDEXES, ALL_IND_COLUMNS

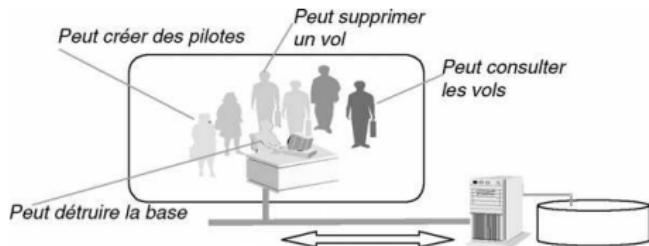
Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

Contrôles d'accès aux données

Système multi-utilisateurs: Généralités

- Identification de l'utilisateur avant toute utilisation de ressource
- Contrôle de l'accès ⇒ Sécurité et cohérence
- Figure: exemple de classification d'un groupe d'utilisateurs



Politiques de contrôle d'accès

- Trois modèles principaux de contrôle des accès dans les SI:
 - Politique de contrôle discrétionnaire (DAC)
 - Politique de contrôle à base de rôles (RBAC)
 - Politique de contrôle obligatoire (MAC)

Contrôles d'accès aux données

Politiques de contrôle d'accès

- Politique de contrôle discrétionnaire:
 - Privilèges spécifiques de chaque utilisateur sur chaque objet
- Politique de contrôle à base de rôles
 - Privilèges spécifiques à un rôle correspondant à une fonction dans l'institution utilisant le Système d'Information.
 - Utilisateurs rattachés à un (des) rôle(s) particulier(s).
- Politique de contrôle obligatoire
 - Niveaux de **classification** des objets & Niveaux d'**habilitation** des utilisateurs
 - **Exemple:**

U peut accéder à l'objet 0

Si Niveau habilit. U > Niveau classif 0

U peut modifier un 0 si

Contrôles d'accès aux données

Norme SQL (2008)

- Autorise la gestion de privilèges
 - Contrôle d'accès discrétionnaire
 - Contrôle d'accès à base de rôles
- Ordres **LCD** (Langage de Contrôle de Données)

```
GRANT [...] TO utilisateur | role
```

```
REVOKE [...] FROM utilisateur | role
```

```
CREATE ROLE nom
```

```
...
```

- Standard SQL assez général dans la gestion de privilèges
 - Extensions / Spécificités propres à chaque SGBD.
 - Cas d'Oracle

Gestion des utilisateurs

Utilisateur

- Identifié au niveau de la base par son nom.
- **Schéma:** collection nommée (du nom de l'utilisateur qui en est propriétaire) d'objets: [nom_Schema] .nom_table

Classification

- Le DBA (*DataBase Administrator*). Il en existe au moins un.
Installation/MàJ BD et outils; Gestion espace disque et espaces pour les données (tablespaces); Gestion des utilisateurs et de leurs objets; Optimisation des performances ; Sauvegardes, restaurations et archivages;
- L'administrateur réseaux (peut être le DBA). Config. Oracle Net (Clt)
- Les développeurs: conception et MàJ de leurs objets.
- Les administrateurs d'applications: Gest. données des applications.
- Les utilisateurs "simples"

Ce sont tous des utilisateurs (au sens Oracle) avec des priviléges différents.

Gestion des utilisateurs

Création d'un utilisateur

- Il faut posséder le privilège CREATE USER
- Syntaxe:

```
CREATE USER utilisateur IDENTIFIED
  { BY motdePasse | EXTERNALLY | GLOBALLY AS 'nomExterne' }
  [ DEFAULT TABLESPACE nomTablespace
  [QUOTA { entier [ K | M ] | UNLIMITED } ON nomTablespace ] ]
  [TEMPORARY TABLESPACE nomTablespace
  [QUOTA { entier [ K | M ] | UNLIMITED } ON nomTablespace ].]
  [PROFILE nomProfil ] [PASSWORD EXPIRE ] [ ACCOUNT { LOCK | UNLOCK } ] ;
```

EXTERNALLY: utiliser l'authentification du SE pour s'identifier à Oracle
 GLOBALLY : se servir de l'authentification d'un annuaire

DEFAULT TABLESPACE ... lui associe un espace disque de travail (tablespace).
 Ex: "USERS", "TEMP", etc. Tablespace "SYSTEM" par défaut

QUOTA : permet de limiter ou pas chaque espace alloué.

PROFILE ...: lui affecte un profil (caract. syst. relat. au CPU et connex.)
 Par défaut, le profil "DEFAULT" est affecté

PASSWORD EXPIRE: pour l'obliger à changer son mdp à la 1ère connexion.

ACCOUNT: pour verrouiller ou libérer l'accès à la base (par défaut UNLOCK)

Gestion des utilisateurs

Création d'un utilisateur - Exemple

```
CREATE USER Paul  
IDENTIFIED BY paul-pwd@orcl1!  
DEFAULT TABLESPACE USERS  
QUOTA 10M ON USERS  
TEMPORARY TABLESPACE TEMP  
QUOTA 5M ON TEMP  
PASSWORD EXPIRE;
```

Paul est déclaré 'utilisateur', ses objets (pas plus de 10 mégaoctets) seront stockés dans USERS. Certaines de ses opérations nécessiteront de ranger des données dans TEMP (pas plus de 5 mégaoctets). Il devra changer son mot de passe à la première connexion. Par défaut, n'a aucun droit sur la BD.

Quelques utilisateurs connus - installés par défaut

- SYS (mot de passe par défaut : CHANGE_ON_INSTALL, rôle DBA)
 - Propriétaire des tables du dictionnaire de données.
 - Il est préférable de ne jamais se connecter sous SYS
- SYSTEM (mot de passe par défaut : MANAGER, rôle DBA).
 - C'est le DBA qu'Oracle vous offre
 - Peut être utilisé pour retransmettre le privilège système à un autre

Il ne faut jamais utiliser leurs *tablespaces* pour les besoins utilisateurs !

Gestion des utilisateurs

Modification d'un utilisateur

- Syntaxe qui reprend les options étudiées dans la création

```
ALTER USER utilisateur IDENTIFIED ...
```

```
...
```

```
...
```

```
[ DEFAULT ROLE { rôle1 [,rôle2 ..] | ALL [EXCEPT rôle1 [,rôle2 ..] | NONE ] }
```

DEFAULT ROLE: lui affecte des rôles (ensembles de priviléges)

Suppression d'un utilisateur

- Syntaxe:

```
DROP USER utilisateur [CASCADE];
```

Option CASCADE: force la suppression et détruit tous les objets du schéma

Dictionnaire

- ALL_USERS, USER_USERS, DBA_USERS

Privilèges

Définition

- C'est un droit:
 - D'exécuter un type d'ordre SQL: privilège système.
 - CREATE USER, CREATE TABLESPACE, ALTER TABLE etc...
 - Les privilèges système diffèrent sensiblement d'un SGBD à un autre.
 - D'accéder à un certain objet d'un autre schéma: privilège objet.
 - SELECT, UPDATE, INSERT etc...
 - Par défaut, un utilisateur a tous les privilèges objet sur ses objets
- Instructions GRANT et REVOKE.
 - Mention ANY: extension du droit à tous les schémas (sauf SYS)
 - Exemple:

Privilège "CREATE ANY TABLE": permet de créer des tables dans tout schéma

Privilège "CREATE TABLE" : ne permet de créer des tables que dans son propre schéma

Privilèges système

Attribution

```

GRANT { privilègeSystème | nomRôle | ALL PRIVILEGES }
       [, { privilègeSystème | nomRôle | ALL PRIVILEGES }]...
TO { utilisateur | nomRôle | PUBLIC }
      [, { utilisateur | nomRôle | PUBLIC }]...
[ WITH ADMIN OPTION ] ;

# WITH ADMIN OPTION : permet d'attribuer aux bénéficiaires le droit de
# retransmettre le(s) privilège(s) reçu(s) à une tierce
# personne (utilisateur(s) ou rôle(s)).

```

Révocation

```

# Détenir au préalable ce privilège avec l'option WITH ADMIN OPTION !!

REVOKE { privilègeSystème | nomRôle | ALL PRIVILEGES }
        [, { privilègeSystème | nomRôle }]...
FROM { utilisateur | nomRôle | PUBLIC } [, { utilisateur | nomRôle }]... ;

# ALL PRIVILEGES: valable si l'user (ou rôle) a tous les privilèges système.
# PUBLIC          : pour annuler le(s) privilège(s) à ceux l'ayant reçu par
#                   l'option PUBLIC.

```

Privilèges système - Exemples

```
GRANT CREATE SESSION,  
CREATE SEQUENCE TO Paul;
```

Paul peut se connecter à la base par un outil (la console par exemple) ou par un programme. Il peut créer des séquences.

```
GRANT CREATE TABLE TO Paul  
WITH ADMIN OPTION ;
```

Paul peut créer des tables dans son schéma et peut retransmettre ce privilège à un tiers.

```
GRANT CREATE SESSION,  
CREATE ANY TABLE,  
DROP ANY TABLE TO Paul2;
```

Paul2 peut se connecter à la base, créer et détruire des tables dans tout schéma.

```
REVOKE CREATE SESSION  
FROM Paul, Paul2 ;
```

Paul et Paul2 ne peuvent plus se connecter à la base.

```
REVOKE ALL PRIVILEGES  
FROM Paul2;
```

Commande incorrecte car Paul2 n'a pas reçu tous les privilèges système.

Privilèges objet

Attribution

```

GRANT { privilègeObjet | nomRôle | ALL PRIVILEGES }
      [(colonne1 [,colonne2]...)] [, { privilègeObjet | nomRôle |
      ALL PRIVILEGES }] [(colonne1 [,colonne2]...)...] ...
ON { [schéma.]nomObjet | { DIRECTORY nomRépertoire | ... [schéma.]nomObjet } } ...
TO { utilisateur | nomRôle | PUBLIC }
      [, { utilisateur | nomRôle | PUBLIC } ]...
[WITH GRANT OPTION] ;

# WITH GRANT OPTION : permet d'attribuer aux bénéficiaires le droit de
                      retransmettre le(s) privilège(s) reçu(s) à une tierce
                      personne (utilisateur(s) ou rôle(s)).

```

Révocation

```

REVOKE { privilègeObjet | ALL PRIVILEGES } [(colonne1 [,colonne2]...)]
      [, { privilègeObjet | ALL PRIVILEGES }] [(colonne1 [,colonne2]...)...] ...
ON { [schéma.]nomObjet | ... [schéma.]nomObjet } }
FROM { utilisateur | nomRôle | PUBLIC } [, { utilisateur | nomRôle | PUBLIC } ]...
[CASCADE CONSTRAINTS] ...;

```

Privilèges objet - Exemples

```
GRANT REFERENCES(brevet),  
       UPDATE(nom, age),  
       SELECT  
ON Pilote  
TO Paul;
```

Affectation des privilèges de lecture de la table Pilote, de modification des colonnes nom et age et de référence à la clé primaire brevet à l'utilisateur Paul

REFERENCES => contrainte d'intégrité tables de schémas distincts.

```
REVOKE UPDATE,  
       SELECT  
ON Pilote  
FROM Paul;
```

Paul ne peut plus modifier ni lire la table Pilote d'un autre utilisateur.

```
REVOKE REFERENCES  
ON Pilote  
FROM Paul  
CASCADE CONSTRAINTS ;
```

Paul ne peut plus bénéficier de la table Pilote pour créer une contrainte référentielle via une clé étrangère.

Quelques privilèges prédéfinis

... pour faciliter la gestion des droits

Nom	Privilèges
GRANT ANY PRIVILEGE	Autorisation de donner tout privilège système.
GRANT ANY OBJECT PRIVILEGE	Autorisation de donner tout privilège objet.
COMMENT ANY TABLE	Commenter une table, vue ou colonne de tout schéma.
SELECT ANY DICTIONARY	Interroger les objets du dictionnaire des données (schéma SYS).
SYSDBA	ALTER DATABASE OPEN MOUNT BACKUP, CREATE DATABASE, ARCHIVELOG, RECOVERY, CREATE SPFILE, RESTRICTED SESSION
SYSOPER	Idem sauf CREATE DATABASE

Donner et reprendre la possibilité d'autoriser tout privilège à l'user

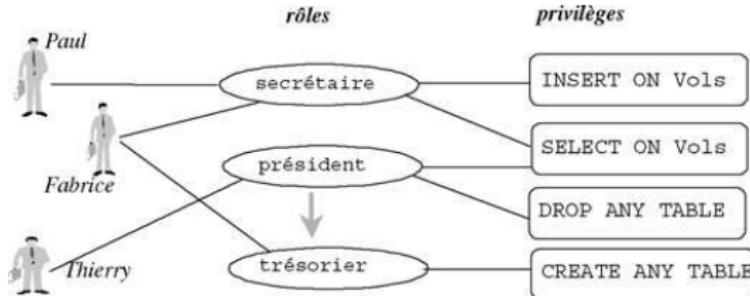
```
GRANT      GRANT ANY OBJECT PRIVILEGE,
           GRANT ANY PRIVILEGE
TO Paul;
```

```
REVOKE    GRANT ANY OBJECT PRIVILEGE,
           GRANT ANY PRIVILEGE,
FROM Paul;
```

Rôles

Définition

- Ensemble nommé de privilèges (système ou objets)
- Accordé à un ou plusieurs utilisateurs, voire à tous (PUBLIC).
- Peut être aussi attribué à un autre rôle pour davantage de droits



```
# Rôle "président": - Privilège objet SELECT sur "Vols"
                    - Privilège système DROP de tables de tout schéma.
                    - Hérite des privilèges du rôle "trésorier".
```

Rôles

Création

- Posséder le privilège CREATE ROLE.

```
CREATE ROLE nomRôle
    [ NOT IDENTIFIED | IDENTIFIED { BY motdePasse |
        USING [schéma.]paquetage | EXTERNALLY | GLOBALLY } ] ;

# NOT IDENTIFIED: utilisation du rôle autorisée sans mot de passe
# IDENTIFIED      : L'utilisateur doit être autorisé par une méthode
```

Exemples

```
CREATE ROLE Voir_Base NOT IDENTIFIED;
```

```
GRANT SELECT ON paul.Pilote
TO Voir_Base
```

```
GRANT Voir_Base
TO Modif_Pilotes      # Modif_Pilotes est un autre rôle!
```

Rôles

Rôles prédefinis

- Rôles prédefinis attribués aux utilisateurs SYSTEM et SYS.
CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE, etc...

Révocation

- REVOKE nomRôle ...

Modification

- ALTER ROLE ...

Suppression

- DROP ROLE ...

Privilèges dans le Dictionnaire

- ALL_TAB_PRIVS, USER_TAB_PRIVS, ALL_COL_PRIVS, USER_COL_PRIVS
- ALL_ROLE_PRIVS, USER_ROLE_PRIVS, DBA_ROLE_PRIVS, ...

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

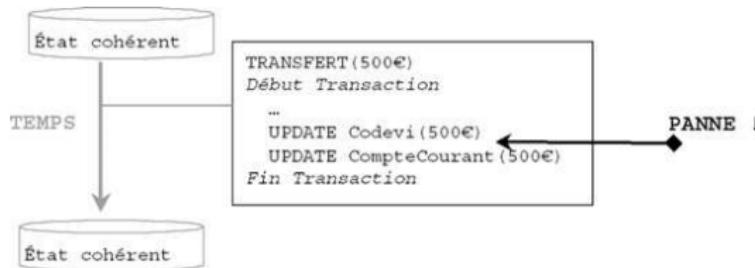
Transactions

Définition

- Bloc d'instructions LMD faisant passer la base de données d'un état cohérent à un autre état cohérent.
- Unité logique de traitement correspondant à un ensemble d'actions successives sur la base de données

En cas de problème logiciel ou matériel

- Aucune des instructions contenues dans la transaction n'est effectuée



Transactions

Principe ACID:

- **Atomicité:** les transactions sont atomiques (tout ou rien)
- **Cohérence:** les transactions maintiennent la cohérence de la base
- **Isolation:** les transactions sont isolées^a les unes des autres
- **Durabilité:** les M&J perdurent même en cas de panne après validation

^aMême pour les transactions concurrentes. Notion de lecture consistante

Mise en œuvre

- Segments d'annulation ou segments **UNDO**
 - Transaction réussie ⇒ BD dans un nouvel état cohérent
 - Échec ⇒ Modifications annulées. BD dans l'état cohérent antérieur.
- ⇒ Seuls les ordres SQL qui modifient des données interviennent
- INSERT, UPDATE et DELETE. Mais pas SELECT

Transactions

Début et fin d'une transaction

- Début **implicite**^a
 - Début de toute commande SQL en début de session.
 - Fin de la transaction précédente.
- Fin **explicite**
 - Commandes SQL de **validation** (COMMIT) ou d'**annulation** (ROLLBACK)
- Fin **implicite**
 - Commandes SQL du LDD ou LCD (CREATE, ALTER, DROP, GRANT, ...)
 - Fin normale/anormale de session. Problème/Arrêt anormal du SGBD.

^aPas d'ordre SQL ou PL/SQL qui marque le début d'une transaction

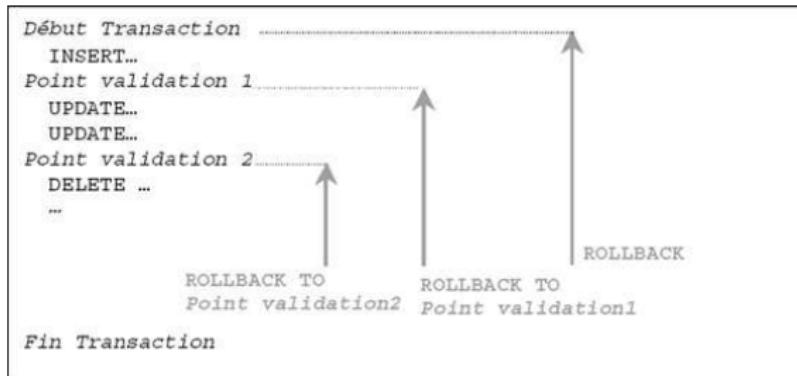
Événement	Validité
COMMIT	Transaction validée.
Commande SQL (LDD ou LCD)	
Fin normale d'une session.	
ROLLBACK	Transaction non validée.
Fin anormale d'une session.	

Contrôle des transactions

Les *savepoints*

- Points de validation. Annulation partielle d'une transaction.
- Positionnement d'un point intermédiaire dans la transaction:
`SAVEPOINT <point_repere> ;`
- Annulation des actions depuis ce point, et non depuis le début.
`ROLLBACK TO <point_repere> ;`

Principe



Contrôle des transactions

Les *savepoints* - Exercice

```
DELETE FROM personne WHERE .... ;  
INSERT INTO TABLE personne VALUES ... ;  
COMMIT ;  
ALTER TABLE societe ADD .... ;  
UPDATE personne ... SET ;  
SAVEPOINT a ;  
DELETE FROM personne WHERE .... ;  
UPDATE personne SET .... ;
```

?

?

**ROLLBACK TO a ;****ROLLBACK** ;

Segment d'annulation

- Chaque base en abrite un ou plusieurs.
- Contient les anciennes valeurs des enregistrements en cours de modification dans les transactions
 - Lecture consistante, annulation de transaction et restauration.
- Pas d'accès ni pour les utilisateurs, ni pour l'administrateur
 - Accès exclusif par le système Oracle. Propriété exclusive de SYS

Principe

- Gère les transactions de la BD.
- Comporte: numéro de fichier, ID du bloc des données modifiées, bloc dans son état initial ⇒ Éléments suffisants pour l'annulation.

Segment UNDO / segment ROLLBACK

- *ROLLBACK segment*: vers. antérieures d'Oracle. Gestion complexe pour l'Admin: création *rollback seg & tablespaces*, assignations, ...
- *UNDO segment*: à partir de la version *9i*. Nombre et création de segments gérés par Oracle lui-même.

Accès concurrents - Lectures consistantes

1/3

Accès concurrent

- Accès simultané de plusieurs utilisateurs à la même donnée

Lecture consistante

- Vue par Oracle 10g
 - Pour un ordre SQL, du début à la fin, pas de changement de valeur pour les données manipulées ou interrogées.
 - Lecture non bloquée par des utilisateurs modifiant les mêmes données.
 - Modification non bloquée par des utilisateurs lisant ces données.
 - Impossible pour un utilisateur de lire les données modifiées par un autre, si elles n'ont pas été validées. Accès à l'état validé précédent.
 - Il faut patienter pour modifier des données en cours de modification dans une autre transaction
- Gestion des lectures et mises à jour
 - Conservation des données mises à jour dans les segments de données
 - Conservation des anciennes valeur dans les segments UNDO.

Accès concurrents - Lectures consistantes

2/3

Principe de la lecture consistante: illustration

1) Toutes les transactions lisent les données de la table

COLONNE A	COLONNE B
PIERRE	10
PAUL	20
JACQUES	30
JEAN	40
ARTHUR	50

2) La transaction A modifie un enregistrement. L'ancienne valeur est placée dans un segment d'annulation UNDO.

COLONNE A	COLONNE B
PIERRE	10
PAUL	20
JACQUES	60
JEAN	40
ARTHUR	50

Segment
UNDO

3) La transaction A lit la donnée modifiée, la transaction B lit l'ancienne valeur placée dans le segment d'annulation UNDO.

A	B
COLONNE A	COLONNE B
PIERRE	10
PAUL	20
JACQUES	60
JEAN	40
ARTHUR	50

- Et si on souhaite effectuer de nombreux ordres SQL sans que les informations apparaissent modifiées entre le début du premier ordre et la fin du dernier ?
 - Transactions en lecture seule (READ ONLY)

Accès concurrents - Lectures consistantes

3/3

Transactions en lecture seule

```
SQL> -- début de la transaction en mode lecture seule
SQL> -- SET TRANSACTION doit être le premier ordre de la transaction
SQL> COMMIT ; -- commencer par un état stable
SQL> SET TRANSACTION READ ONLY ;
SQL>
SQL> SELECT * FROM tableA ;
...
SQL> SELECT * FROM tableB ;
...
SQL> COMMIT ; -- indique la fin du " read-only "
```

- Les données seront comme figées, telles qu'elles étaient au moment du "top chrono" donné par le "read-only"
- Introduction d'une faible baisse de performance: une plus grande utilisation des segments d'annulation due aux transactions qui continuent de modifier les données.

Accès concurrents - La pose de verrous

1/2

Mécanisme de verrouillage

- Réserver un objet à un utilisateur et à empêcher les autres de le modifier, jusqu'à l'exécution d'un ordre COMMIT ou ROLLBACK.

Verrouillage implicite des données

- Gestion des verrous transparente pour l'utilisateur.
- Granularité fine : on peut ne verrouiller qu'une table ou un tuple.

Mise à jour des données (INSERT, UPDATE et DELETE)

- Verrou automatique sur le tuple (ROW) de la table concernée.
- Création d'un segment UNDO: modifications visibles du seul user

Consultation de données (SELECT)

- Aucun verrou.
- Voit les données dans l'état où elles étaient au début de l'ordre

Accès concurrents - La pose de verrous

2/2

- Il est possible d'utiliser des ordres SQL pour poser soi-même des verrous sur des données ou des tables.
⇒ Verrouillage explicite

Verrouillage explicite

- Verrouillage de table:

```
LOCK TABLE <nom_table> IN {EXCLUSIVE | SHARED } MODE
```

- Verrouillage de tuple:

```
LOCK TABLE <nom_table> IN {ROW EXCLUSIVE | ROW SHARED |  
SHARE ROW EXCLUSIVE} MODE
```

- Il est fortement déconseillé d'utiliser LOCK TABLE
 - Verrouillage automatique plus performant et plus simple que la gestion manuelle
- Dictionnaire: V\$LOCK, GV\$LOCKED_OBJECT, %LOCK% ...

Plan

- 1 Introduction Générale
- 2 Rappels sur le LDD/LMD SQL - Interfaces conversationnelles
- 3 Dictionnaire Oracle - Importation de données - Contraintes
- 4 Objets avancés
- 5 Contrôles d'accès
- 6 Transactions
- 7 PL/SQL

PL/SQL - Généralités

PL/SQL

- *Procedural Language / Structured Query Language*
- Langage propre à Oracle
- Extension de SQL:
 - Ajout de mécanismes pour parcourir les résultats (curseurs), traiter les exceptions et réagir à l'état de la base (déclencheurs - triggers)
 - Cohabitation Structures de contrôle (IF, FOR et WHILE) / Instructions SQL (SELECT, INSERT, UPDATE et DELETE).
- ⇒ Étend l'expressivité de SQL pour de nouveaux types de contraintes

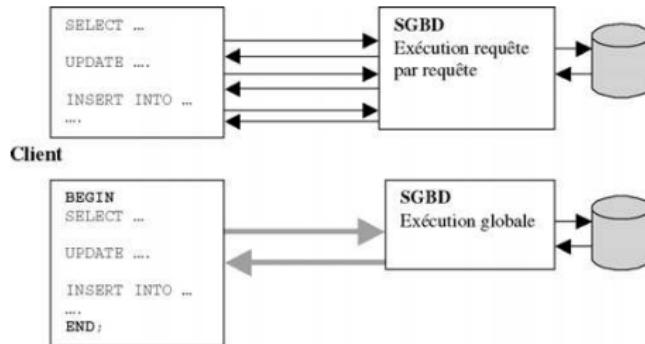
Comparaison avec les autres bases de données

- PL/pgSQL de PostgreSQL
- Transact-SQL de Microsoft SQL Server et Sybase
- SQL/PSM de MySQL

PL/SQL

Environnement client-serveur

- Désencombrement du trafic réseau
 - Un bloc de programmation plutôt que plusieurs ordres SQL
 - Résultats intermédiaires traités côté serveur
 - Seul le résultat final est retourné au client.
 - Diminution des échanges réseaux
 - Augmentation des performances globales de vos applications.



PL/SQL

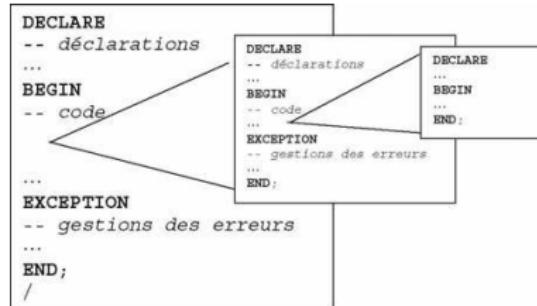
Autres Avantages

- Facilité de programmation
 - Langage simple d'apprentissage et de mise en œuvre
 - Offre une grande lisibilité en phase de maintenance
- Support de la POO
 - Types objet proposant une approche aisée et puissante de la prog. objet
 - Modularité: blocs nommés (procédure, fonction) ⇒ Réutilisabilité.
- Portabilité:
 - Indépendance par rapport au S.E. qui héberge le serveur Oracle
 - En changeant de système, les applicatifs n'ont pas à être modifiés.
- Intégration avec les données des tables:
 - Types de données et instructions disponibles sous SQL
 - Mécanismes: Parcourir des résultats de requêtes (cursor). Traiter des erreurs (exceptions). Manipuler des données complexes (paquetages DBMS_xxx). Programmer des transactions.

PL/SQL - Structure d'un programme

Programme PL/SQL (pas nommé) - Bloc : 3 sections

- **DECLARE** (section optionnelle): déclare les variables, types, curseurs, exceptions, etc. ;
- **BEGIN** (section obligatoire): contient le code PL/SQL, jusqu'à l'instruction **END**. Le caractère "**/**" termine un bloc dans SQL*Plus.
- **EXCEPTION** (section optionnelle): traite les erreurs renvoyées.



- Encapsulation possible de sous-blocs. Attention: portée des variables.

PL/SQL - Syntaxe

- Jeu de caractères - *not case sensitive*

[a-zA-Z0-9] ()+-*/<>!=~^;:.'@%, "#\$&_|\{\}?[] ; ↲ ↳ ↲

- Identificateurs - Une lettre suivie de symboles (taille max=30)

:) X, t2, téléphone#, code_brevet, codeBrevet, oracle\$nombre
 :-(toi&moi, debit-credit, on/off, code\brevet

- Commentaires

- Monolignes: - - Ceci est un commentaire sur une ligne.
- Multilignes: /* Ceci peut être un commentaire sur plusieurs ligne. */

Variables (zone DECLARE du bloc)

PL/SQL est capable de manipuler des variables et des constantes.

- Variables PL/SQL: Scalaires (une seule valeur d'un type SQL); Composites (%ROWTYPE, %TYPE et RECORD); Références (REF) ; LOB.
- Variables non PL/SQL: Définies sous SQL*Plus (de substitution et globales). Variables hôtes (déclarées dans des prog. précompilés).

PL/SQL - Variables

- Variables scalaires: **Déclaration**

```
identificateur [CONSTANT] typeDeDonnée [NOT NULL] [:= |  
DEFAULT expression];
```

Exemple

```
DECLARE  
    c_euro CONSTANT NUMBER := 6.569;  
    prix NUMBER; /* prix initialisé à NULL */  
    date_courante DATE DEFAULT SYSDATE;
```

- Affectation:** 3 possibilités (:= ; DEFAULT ; INTO).

Exemple

```
DECLARE  
    v_prenom      VARCHAR2(30) NOT NULL DEFAULT 'UNKNOWN';  
    v_dateRecr   DATE := SYSDATE + 2;  
BEGIN  
    SELECT prenom INTO v_prenom FROM personne WHERE nom = 'Wade';  
    ...
```

PL/SQL - Types

- Type **%TYPE**
 - Rattachement du type à celle d'un attribut de table ou vue
nom_variable nom_table.nom_attribut %TYPE
 - Rattachement du type de la variable à une autre variable déjà déclarée
nom_new_var nom_var %TYPE
- Type enregistrement - ligne - **%ROWTYPE**
 - Description d'un enregistrement unique complet

Déclaration: nom_variable nom_table %ROWTYPE
Utilisation: nom_variable.nom_d_un_des_champs

Exemple

```

DECLARE
    c_euro      CONSTANT NUMBER := 6.569;
    prix_euro   table_produits.prix %TYPE ;
    prix_fr     prix_euro %TYPE := prix_euro * c_euro ;
    produits    table_produits %ROWTYPE;
BEGIN
    SELECT * INTO produits from Table_Produits WHERE id=1;
    prix_euro := produits.prix;

```

PL/SQL - Types personnalisés - RECORD & TABLE

RECORD

- Structure de donnée personnalisée

```
TYPE nomRecord IS RECORD ( nomChamp typeDonnées [[NOT
NULL] := | DEFAULT expression] [,nomChamp .. ]...);
```

- Un champ d'un RECORD peut lui même être un autre RECORD.

Exemple

```
DECLARE
    TYPE avionAirbus_rec IS RECORD (nserie CHAR(10), nomAvion CHAR(20),
                                      usine CHAR(10) := 'Blagnac', nbHVol NUMBER(7,2));
    r_unA320 avionAirbus_rec;
    r_FGLFS  avionAirbus_rec;          -- Déclaration de variables
BEGIN
    r_unA320.nserie    := 'A1';        -- Initialisation
    r_unA320.nomAvion := 'A320-200';
    r_unA320.nbHVol   := 2500.60;
    r_FGLFS           := r_unA320;     -- Affectation d'un RECORD.
```

PL/SQL - Types personnalisés - RECORD & TABLE

TABLE - (1/3)

- Tableau dynamique: défini sans dimension prédéfinie
 - Clé primaire de type BINARY_INTEGER : -2147483647 ↔ 2147483647.
 - Colonne de type scalaire, %TYPE, %ROWTYPE ou RECORD)

```
TYPE nomTypeTableau IS TABLE OF typeScalaire |
variable%TYPE | table.colonne%TYPE [NOT NULL] |
table.%ROWTYPE [INDEX BY BINARY_INTEGER];
```

- [INDEX BY BINARY_INTEGER] facultatif depuis PL/SQL version 8.
 - Absence: table considéré comme une *nested table* (extension objet).

PL/SQL - Types personnalisés - RECORD & TABLE

TABLE - (2/3)

Exemple

```
DECLARE
    TYPE nomPilotes_tytab IS TABLE OF Pilote.nom%TYPE
                                INDEX BY BINARY_INTEGER;

    TYPE pilotes_tytab      IS TABLE OF Pilote%ROWTYPE
                                INDEX BY BINARY_INTEGER;

    tab_nomPilotes nomPilotes_tytab;
    tab_pilotes     pilotes_tytab;

BEGIN
    tab_nomPilotes(7800)    := 'Bidal';
    tab_nomPilotes(-2)      := 'Pascal';
    tab_nomPilotes(-1)      := 'Moussa';
    tab_pilotes(0).brevet := 'PL-0';
END;
```

PL/SQL - Types personnalisés - RECORD & TABLE

TABLE - (3/3)

- Fonctions et procédures pour manipuler les tableaux

EXISTS (<i>x</i>)	Retourne TRUE si le <i>x</i> ^e élément du tableau existe.
COUNT	Retourne le nombre d'éléments du tableau.
FIRST / LAST	Retourne le premier/dernier indice du tableau (NULL si tableau vide)
PRIOR (<i>x</i>) / NEXT (<i>x</i>)	Retourne l'élément avant/après le <i>x</i> ^e élément du tableau.
DELETE	Supprime un ou plusieurs éléments au tableau.
DELETE (<i>x</i>)	
DELETE (<i>x, y</i>)	

Exemple (suite exemple diapo précédente)

```
IF tab_pilotes.EXISTS(0) THEN ... ==> VRAI (il y a un élé à l'indice 0)
v_nombre := tab_nomPilotes.COUNT; ==> La variable v_nombre contient 3.
v_premier := tab_brevets.FIRST;    ==> v_premier contient -2.
v_dernier := tab_brevets.LAST;     ==> v_dernier contient 7800.
```

PL/SQL - Variables

Résolution de noms

- En cas de conflit de noms (variables ou colonnes):
 - Le nom de la colonne de la table est **prioritaire**

Exemple:

```

DECLARE
    nom CHAR(20) := 'Pierre Lamothe';
BEGIN
    DELETE FROM Pilote WHERE nom = nom; -- supprime tous les pilotes !!
    ...
  
```

Solutions:

Préfixer les variables	Étiquette de bloc
<pre> DECLARE v_nom CHAR(20) := 'Pierre Lamothe'; BEGIN DELETE FROM Pilote WHERE nom = v_nom; END; </pre>	<pre> <<principal>> DECLARE nom CHAR(20) := 'Pierre Lamothe'; BEGIN DELETE FROM Pilote WHERE nom = principal.nom; END; </pre>

PL/SQL - Variable de substitution & Variable de session

Variables de substitution

- Passage de paramètres en entrée d'un bloc PL/SQL
- Identification du paramètre par le symbole **&** en préfixe au nom
- Pas de déclaration de la variable à faire (partie DECLARE)
- Directive SQL*Plus ACCEPT si on veut afficher un message d'invite
 - Sinon, SQL*Plus demandera simplement la valeur du paramètre

```
ACCEPT s_number PROMPT 'Entrez le numero du vol : '
ACCEPT s_name   PROMPT 'Entrez le nom    du vol : '
```

DECLARE

```
  TYPE type_vol is RECORD (num NUMBER, nom VARCHAR2(20));
  vol_particulier type_vol;
BEGIN
  vol_particulier.num := &s_number ;
  vol_particulier.nom := '&s_name' ; -- Il y aurait ERREUR sans les ''
...
/* Fichier ex.plsql */ ==> Sous SQL*Plus: SQL> [start | @] ex.plsql
```

PL/SQL - Variable de substitution & Variable de session

Variables de session

- Variables globale (de session) définies sous SQL*Plus
- Directive SQL*Plus à utiliser avant le bloc est VARIABLE
- Dans le code PL/SQL, il faut faire préfixer le nom du symbole :
- Affichage de la variable sous SQL*Plus: directive PRINT.

```
SQL> VARIABLE g_compteur
SQL> DECLARE
 2   v_compteur NUMBER(3) := 99;
 3   BEGIN
 4     :g_compteur := v_compteur + 1 ;
 5   END;
 6 /
PL/SQL procedure successfully completed.
```

```
SQL> PRINT g_compteur
```

```
G_COMPTEUR
```

```
-----
```

```
100
```

PL/SQL - Conventions recommandées

- Lisibilité et maintenabilité du code PL/SQL

Objet	Convention	Exemple
Variable	v_nomVariable	v_compteur
Constante	c_nomConstante	c_pi
Exception	e_nomException	e_pasTrouvé
Type RECORD	nomRecord_rec	pilote_rec
Variable RECORD	r_nomVariable	r_pilote
Variable ROWTYPE	rty_nomVariable	rty_pilote
Type-tableau	nomTypeTableau_tytab	pilotes_tytab
Variable tableau	tab_nomTableau	tab_pilotes
Curseur	nomCurseur_cur	pilotes_cur
Variable de substitution (SQL*Plus)	s_nomVariable	s_brevet
Variable de session (globale)	g_nomVariable	g_brevet

PL/SQL - Structures de contrôles

- Structure conditionnelle IF

IF-THEN	IF-THEN-ELSE	IF-THEN-ELSIF
IF condition THEN instructions; END IF;	IF condition THEN instructions; ELSE instructions; END IF;	IF condition1 THEN instructions; ELSIF condition2 THEN instructions; ELSE instructions; END IF;

```

DECLARE
    v_téléphone CHAR(12) NOT NULL := '77-802-55-09';
BEGIN
    IF SUBSTR(v_téléphone,1,2)='77' THEN
        DBMS_OUTPUT.PUT_LINE ('C''est une ligne Orange !');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('C''est une autre ligne ou un fixe ... ');
    END IF;
END;

```

PL/SQL - Structures de contrôles

• Structure CASE

CASE variable		CASE
WHEN expr1 THEN instruc1;		WHEN condition1 THEN instruct1;
WHEN expr2 THEN instruc2;		WHEN condition2 THEN instruct2;
...		...
WHEN exprN THEN instrucN;		WHEN conditionN THEN instructN;
[ELSE instrucN+1;]		[ELSE instrucN+1;]
END CASE		END CASE

Example

```
CASE v_grade
    WHEN 'A' THEN dbms_output.put_line('Excellent');
    WHEN 'B' THEN dbms_output.put_line('Very Good');
    WHEN 'C' THEN dbms_output.put_line('Good');
    WHEN 'D' THEN dbms_output.put_line('Fair');
    WHEN 'F' THEN dbms_output.put_line('Poor');
    ELSE dbms_output.put_line('No such grade');
END CASE;
```

PL/SQL - Structures de contrôles

- Structure **WHILE**

```
WHILE condition LOOP  
    instructions;  
END LOOP;
```

- Structure **LOOP**

```
LOOP  
    instructions;  
    EXIT [WHEN condition;]  
END LOOP;
```

- Structure **FOR**

```
FOR compteur IN [REVERSE] valeurInf..valeurSup LOOP  
    instructions;  
END LOOP;  
    -- Déclaration de "compteur" pas obligatoire  
    -- Il ne faut pas modifier cette variable dans la boucle
```

PL/SQL - Entrées / Sorties

Paquetage DBMS_OUTPUT

- Activation du paquetage sous SQL*PLUS: SET SERVEROUTPUT ON
 - ① Il appelle DBMS_OUTPUT.ENABLE; avec un **buffer** de taille 20000 octets
 - ② Appelle la procédure DBMS_OUTPUT.GET_LINES; et redirige les messages du buffer dans la **sortie standard** .
 - ③ Entre autres ...
- Activation du paquetage sous PL/SQL: DBMS_OUTPUT.ENABLE;
- Utilisation d'une procédure du paquetage: DBMS_OUTPUT.Nom_Proc;

Quelques autres Paquetages

- DBMS_RANDOM: génération de nombres aléatoires
- DBMS_LOCK: gestion des verrous
- DBMS_ROWID: manipulation des *rowids*
- DBMS_SQL: construction statique ou dynamique d'ordres SQL.

PL/SQL - Entrées / Sorties

Procédures disponibles de DBMS_OUTPUT

Procédure	Explication
ENABLE (taille Tampon IN INTEGER DEFAULT 2000)	Activation du paquetage dans la session.
DISABLE	Désactivation du paquetage dans la session.
PUT(ligne IN VARCHAR2 DATE NUMBER);	Mise dans le tampon d'un paramètre.
NEW_LINE(ligne OUT VARCHAR2, statut OUT INTEGER)	Écriture du caractère fin de ligne dans le tampon.
PUT_LINE(ligne IN VARCHAR2 DATE NUMBER);	PUT puis NEW_LINE.
GET_LINE(ligne OUT VARCHAR2, statut OUT INTEGER)	Affectation d'une chaîne du tampon dans une variable.
GET_LINES(tab OUT DBMS_OUTPUT.CHARARR, nombreLignes IN OUT INTEGER); CHARARR table de VARCHAR2(255)	Affectation de chaînes du tampon dans un tableau.

- Rappel:
 - SQL*Plus: activation du paquetage par SET SERVEROUTPUT ON
 - Une fois activé, l'option reste valable durant toute la session SQL*Plus

PL/SQL - Entrées / Sorties

Exemples - (1/4)

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
 2  DBMS_OUTPUT.PUT_LINE ('Ligne 1');
 3  DBMS_OUTPUT.PUT_LINE ('Ligne 2');
 4  DBMS_OUTPUT.PUT_LINE ('Ligne 3');
 5 END;
 6 /
```

Ligne 1

Ligne 2

Ligne 3

PL/SQL procedure successfully completed

PL/SQL - Entrées / Sorties

Exemples - (2/4)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  1   x NUMBER;
  2   BEGIN
  3     DBMS_OUTPUT.PUT_LINE ('Ligne 1');
  4     FOR i IN 1 .. 1000000000 LOOP          -- Boucle pour
  5       x := x;                            -- faire passer le temps !
  6     END LOOP;                          -- (une sorte de SLEEP)
  7     DBMS_OUTPUT.PUT_LINE ('Ligne 2');
  8     DBMS_OUTPUT.PUT_LINE ('Ligne 3');
  9   END;
 10 /
<-- Longue attente -->
Ligne 1
Ligne 2
Ligne 3
```

PL/SQL procedure successfully completed

PL/SQL - Entrées / Sorties

Exemples - (3/4)

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
 1 DBMS_OUTPUT.PUT_LINE ('Ligne 1');
 2 DBMS_OUTPUT.PUT_LINE ('Ligne 2');
 3 DBMS_OUTPUT.PUT_LINE ('Ligne 3');
 4 DBMS_OUTPUT.DISABLE;
 5 END;
 6 /
```

<-- Aucune sortie ! -->

PL/SQL procedure successfully completed

PL/SQL - Entrées / Sorties

Exemples - (4/4)

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  1   lines DBMS_OUTPUT.CHARARR;
  2   num_lines NUMBER := 2;
  3 BEGIN
  4   DBMS_OUTPUT.PUT_LINE ('Ligne 1');
  5   DBMS_OUTPUT.PUT_LINE ('Ligne 2');
  6   DBMS_OUTPUT.PUT_LINE ('Ligne 3');
  7   DBMS_OUTPUT.GET_LINES (lines, num_lines);
  8   num_lines := 1;
  9   DBMS_OUTPUT.GET_LINES (lines, num_lines);
 10  DBMS_OUTPUT.PUT_LINE ('Première ligne du buffer: ' || lines(1));
 11 END;
 12 /
```

Première ligne du buffer: Ligne 3

PL/SQL procedure successfully completed

PL/SQL - Interactions avec la base

Extraire des données

```
SELECT col1[, col2] ... INTO { nomVarPLSQL [, nomVarPLSQL] ... |  
                                nomRECORD } FROM nomTable ...;
```

- Une requête SELECT ... INTO doit renvoyer un seul enregistrement.
 - Traitement des requêtes renvoyant plusieurs enregistrements: **curseurs**
- Sinon
 - Plusieurs résultat ⇒ exception ORA-01422 TOO_MANY_ROWS
 - Aucun résultat ⇒ exception ORA-01403 NO_DATA_FOUND

PL/SQL - Interactions avec la base

Manipuler des données

- **Insertion:**

```
BEGIN  
    INSERT INTO Pilote VALUES ('PL-5', rty_pilote.nom, 500, 'AF');  
    ...
```

- **Modifications:**

```
BEGIN  
    UPDATE Pilote SET nbHVol = nbHVol + 10 WHERE compa = 'AF';  
    dbms_output.put_line('NbPiloteModif = ' || SQL%ROWCOUNT);  
    ...
```

- **Suppressions:**

```
BEGIN  
    DELETE FROM Pilote WHERE nbHVol < 100 ;  
    dbms_output.put_line('NbPiloteSuppr = ' || SQL%ROWCOUNT);  
    ...
```

PL/SQL - Interactions avec la base

Curseurs **implicites**

- Utilisation d'un curseur implicite^a pour chaque opération du LMD:
INSERT, UPDATE et DELETE
- Nom du curseur: SQL.
Exploitable après exécution de l'instruction
- La commande suivante remplace l'ancien curseur par un nouveau
- Quelques attributs:

SQL%ROWCOUNT Nombre de lignes affectées par la dernière instruction LMD.

SQL%FOUND Booléen valant TRUE si la dernière instruction LMD affecte au moins un enregistrement.

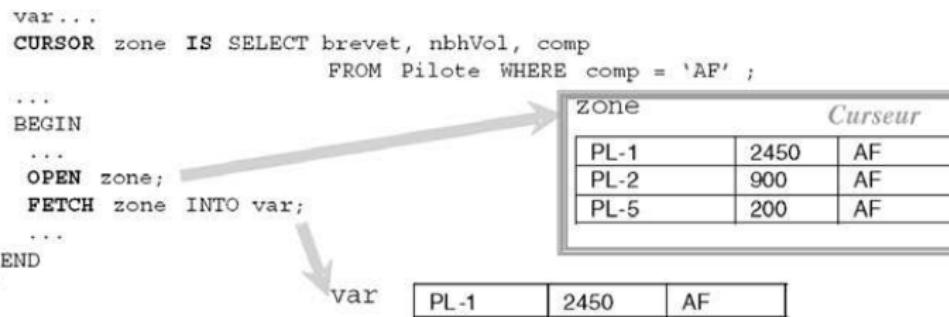
SQL%NOTFOUND Booléen valant TRUE si la dernière instruction LMD n'affecte aucun enregistrement.

^aCurseur **explicite** pour parcourir un ensemble d'enregistrements.

PL/SQL - Curseurs (explicites)

Curseur - Généralités

- Zone mémoire permettant de traiter individuellement chaque ligne renvoyée par un SELECT
 - Resultset (JDBC), RecordSet (ASP), DataSet (.NET)*
- De l'ouverture à la fermeture, il contient en permanence l'adresse de la ligne courante.



- Plusieurs manières de parcourir un curseur.
- Plusieurs types de curseurs à parcourir.

PL/SQL - Curseurs - Instructions

Instruction	Commentaires et exemples
CURSOR nomCurseur IS requête;	Déclaration du curseur. <pre>CURSOR zone1 IS SELECT brevet, nbHVol, comp FROM Pilote WHERE comp = 'AF';</pre>
OPEN nomCurseur;	Ouverture du curseur (chargement des lignes). Aucune exception n'est levée si la requête ne ramène aucune ligne. <pre>OPEN zone1;</pre>
FETCH nomCurseur INTO listeVariables nomRECORD;	Positionnement sur la ligne suivante et chargement de l'enregistrement courant dans une ou plusieurs variables. <pre>FETCH zone1 INTO var1,var2,var3;</pre>
CLOSE nomCurseur;	Ferme le curseur. L'exception INVALID_CURSOR se déclenche si des accès au curseur sont opérés après sa fermeture. <pre>CLOSE zone1;</pre>
nomCurseur%ISOPEN	Retourne TRUE si le curseur est ouvert, FALSE sinon. <pre>IF zone1%ISOPEN THEN ...</pre>
nomCurseur%NOTFOUND	Retourne TRUE si le dernier FETCH n'a pas renvoyé de ligne (fin de curseur). <pre>EXIT WHEN zone1%NOTFOUND;</pre>
nomCurseur%FOUND	Retourne TRUE si le dernier FETCH a renvoyé une ligne. <pre>WHILE (zone1%FOUND) LOOP</pre>
nomCurseur%ROWCOUNT	Retourne le nombre total de lignes traitées jusqu'à présent (pointeur absolu).

PL/SQL - Curseurs

Curseur - Parcours

- Parcours contrôlé: LOOP (Répéter), WHILE (Tant Que)
- Parcours semi-automatique (systématique): FOR (Pour)

Exemple (1/2) - Parcours Contrôlé

Tant que

Répéter

```

DECLARE
    CURSOR zone1 IS SELECT brevet, nbHVol, comp
                  FROM Pilote WHERE comp = 'AF';
    var1    Pilote.brevet%TYPE;
    var2    Pilote.nbHVol%TYPE;
    var3    Pilote.comp%TYPE;
    totalHeures NUMBER := 0;

BEGIN
    OPEN zone1;
    FETCH zone1 INTO var1,var2,var3;
    WHILE (zone1%FOUND) LOOP
        totalHeures := totalHeures + var2;
        FETCH zone1 INTO var1,var2,var3;
    END LOOP;
    CLOSE zone1;
    ...
END;
/
```

```

BEGIN
    OPEN zone1;
    LOOP
        FETCH zone1 INTO var1,var2,var3;
        EXIT WHEN zone1%NOTFOUND;
        totalHeures := totalHeures + var2;
    END LOOP;
    CLOSE zone1;
    ...
END;
/
```

PL/SQL - Curseurs

Exemple (2/2) - Parcours Contrôlé - Utilisation de %ROWTYPE

- Éviter de déclarer autant de variables que de colonnes.
- Accès aux valeurs des colonnes par la notation pointée.

```

DECLARE
    CURSOR zone2 IS SELECT brevet, nom
                  FROM Pilote WHERE NOT (comp='AF');
    enreg zone2%ROWTYPE;

BEGIN
    OPEN zone2;
    FETCH zone2 INTO enreg;
    WHILE (zone2%FOUND) LOOP
        DBMS_OUTPUT.PUT_LINE('nom : ' || enreg.nom ||
                             ' (' || enreg.brevet || ') .');
        FETCH zone2 INTO enreg ;
    END LOOP;
    CLOSE zone2;
END;
/

```

- Il est possible d'utiliser un RECORD conforme (structure) au curseur
- **Rmq:** FETCH ne lève pas d'exception s'il n'y a plus de données.

PL/SQL - Curseurs

Curseur - Boucle FOR (gestion semi-automatique) - (1/2)

- Facilite la programmation:
 - Plus besoin des directives OPEN, FETCH et CLOSE.
 - La boucle s'arrête d'elle-même à la fin de l'extraction des lignes.
 - La variable de réception du curseur est implicitement déclarée comme %ROWTYPE du curseur

```

DECLARE                                     Déclaration du curseur.
CURSOR zone3 IS SELECT brevet, nom
                  FROM Pilote WHERE NOT (comp='AF') ;

BEGIN
  FOR enreg IN zone3 LOOP                 Itération dans le curseur.
    DBMS_OUTPUT.PUT_LINE('nom : ' || enreg.nom ||
                         ' (' || enreg.brevet || ') .');
  END LOOP;
END;
/

```

PL/SQL - Curseurs

Curseur - Boucle FOR - Curseur temporaire (implicite) - (2/2)

- Curseur déclarer à l'intérieur de l'instruction FOR.
- Impossibilité de réutiliser le curseur (existence interne à la boucle)
- Impossibilité d'utiliser des paramètres.

```
BEGIN
  FOR enreg IN
    (SELECT brevet, nom FROM Pilote WHERE
      NOT (comp = 'AF')) LOOP
    DBMS_OUTPUT.PUT_LINE('nom : ' || enreg.nom || '
                          (' || enreg.brevet || ') .');
  END LOOP;
END ;
/
```

PL/SQL - Curseurs et paramètres

Curseurs paramétrés

```

DECLARE
    CURSOR zone5(p_codecomp IN VARCHAR2) IS
        SELECT brevet, nom
            FROM Pilote WHERE comp = p_codecomp;
    enregbis zone5%ROWTYPE;
BEGIN
    FOR enreg IN zone5('AF') LOOP
        DBMS_OUTPUT.PUT_LINE('AF, nom : ' || enreg.nom ||
                            ' (' || enreg.brevet || ') .');
    END LOOP;

    OPEN zone5('SING');
    FETCH zone5 INTO enregbis ;
    WHILE (zone5%FOUND) LOOP
        DBMS_OUTPUT.PUT_LINE('SING, nom : ' ||
                            enregbis.nom|| '('||enregbis.brevet|| ') .');
        FETCH zone5 INTO enregbis ;
    END LOOP;
    CLOSE zone5;
END;
/

```

Déclaration du curseur avec un paramètre.

Chargement et parcours du curseur en passant le paramètre 'AF'.

Chargement et parcours du curseur en passant le paramètre 'SING'.

PL/SQL - Curseurs et Mises à jour (1/2)

FOR UPDATE / CURRENT OF

- FOR UPDATE: pour verrouiller les lignes d'une table interrogée par un curseur dans le but de mettre à jour la table.

```
CURSOR nomCurseur[(paramètres)] IS
    SELECT ... FROM {nomTable | nomVue } WHERE ...
FOR UPDATE [OF [[schéma.] {nomTable | nomVue }.]colonne [, ...]
            [ NOWAIT | WAIT entier ]
```

- OF: Désignation des colonnes à verrouiller. Par défaut tous les champs
- WAIT/NOWAIT: attendre (combien de secondes ?) ou pas que les lignes soient déverrouillées par une autre session.
- Validation (COMMIT) interdite avant la fermeture d'un FOR UPDATE
- Pas de DISTINCT, GROUP BY, agrégat et fonctions ensemblistes

PL/SQL - Curseurs et Mises à jour (2/2)

FOR UPDATE / CURRENT OF

- CURRENT OF: pour pouvoir modifier facilement la ligne courante d'un curseur (UPDATE ou DELETE à répercuter au niveau de la table).
- Clause WHERE CURRENT OF située au niveau de l'instruction de MàJ
- CURRENT OF à utiliser obligatoirement avec FOR UPDATE

DECLARE

```
CURSOR zoneModifiable IS SELECT * FROM Pilote
    FOR UPDATE OF nbHVol NOWAIT;
BEGIN
    FOR enreg IN zoneModifiable LOOP
        IF enreg.comp = 'AF' THEN
            UPDATE Pilote SET nbHVol = nbHVol + 100
                WHERE CURRENT OF zoneModifiable;
        END IF;
    END LOOP;
    COMMIT;
END;
```

PL/SQL - Exceptions

Exemple de session - (1/2)

```
SQL> SELECT * FROM adultes;
```

NUM_H	NOM	AGE
1	DIOP	25
2	FALL	30

```
SQL> DECLARE
 2      v_nom adultes.nom%TYPE;
 3  BEGIN
 4      SELECT nom INTO v_nom FROM adultes;
 5      DBMS_OUTPUT.PUT_LINE('Le nom de l''adulte est: ' || v_nom);
 6  END;
 7 /
```

ERROR at line 1:

ORA-01422: exact fetch returns more than requested number of rows ...

PL/SQL - Exceptions

Exemple de session - (2/2)

```
SQL> DECLARE
 2      v_nom adultes.nom%TYPE;
 3
 4      BEGIN
 5          SELECT nom INTO v_nom FROM adultes;
 6          DBMS_OUTPUT.PUT_LINE('Le nom de l''adulte est: ' || v_nom);
 7
 8      EXCEPTION
 9          WHEN TOO_MANY_ROWS THEN
10              DBMS_OUTPUT.PUT_LINE('Trop de lignes !');
11      END;
12  /
```

Trop de lignes !

PL/SQL procedure successfully completed.

PL/SQL - Exceptions interne et utilisateur

Exceptions internes

- Erreur interne levée par Oracle
 - Un nom d'exception (Ex: TOO_MANY_ROWS), un message d'erreur (variable SQLERRM), un code d'erreur (variable SQLCODE, Ex: -01422)
- Erreur interne non documentée (par de nom d'exception PL/SQL)
 - Association d'un nom au numéro de l'erreur dans la zone DECLARE
`PRAGMA EXCEPTION_INIT (mon_nom_excep, SQLCODE);`

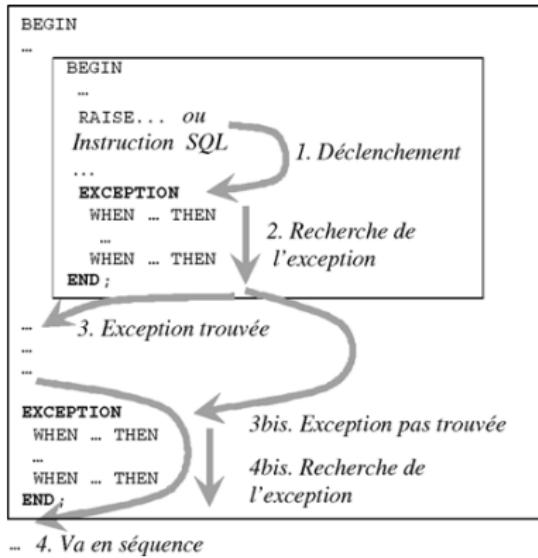
Exceptions utilisateurs

- Erreur définie par l'utilisateur
 - Traiter une erreur applicative comme une erreur interne.
 - Déclaration dans la zone DECLARE
`Nom_exception EXCEPTION;`
- Association d'un code d'erreur à une exception utilisateur
`RAISE_APPLICATION_ERROR (numéro_erreur, message);`

PL/SQL - Exceptions

Traitement des exceptions

```
BEGIN
...
IF (...) THEN RAISE PILOTE TROP JEUNE ;
...
SELECT ... INTO ... FROM ... ;
...
EXCEPTION
  WHEN NO DATA FOUND THEN
    Instructions - A
  WHEN ZERO DIVIDE THEN
    Instructions - B
  WHEN PILOTE TROP JEUNE THEN
    Instructions - C
  WHEN OTHERS THEN
    Instructions - D
END ;
```



- Pas de différence de traitement entre erreurs internes et exceptions posées par l'utilisateur

PL/SQL - Exceptions

Exceptions internes prédéfinies

- Celles qui se produisent le plus souvent.
- Association d'un nom pour le traitement dans le bloc EXCEPTION

Nom de l'exception	Numéro	Commentaires
CURSOR_ALREADY_OPEN	ORA-06511	Ouverture d'un curseur déjà ouvert.
NO_DATA_FOUND	ORA-01403	Requête ne retournant aucun résultat.
TOO_MANY_ROWS	ORA-01422	Requête retournant plusieurs lignes.
ZERO_DIVIDE	ORA-01476	Division par zéro.
...	...	

- Code d'erreur SQLCODE (Ex: -06511, -01403, ...) utilisable sous PL/SQL, Java (JDBC), C etc.
- NO_DATA_FOUND: opérationnel qu'avec SELECT. Pour UPDATE/DELETE utiliser Curseur implicite (SQL%NOTFOUND) + Exception utilisateur

PL/SQL - Exceptions

Exceptions utilisateur

- Possibilité de définir ses propres exceptions
- Traiter une erreur applicative comme une erreur renvoyée par la base
- Déclenchement explicite avec RAISE nomException

```
SQL> DECLARE
  3      exception_aucune_suppression EXCEPTION;          /* Déclaration */
  4      BEGIN
  5          DELETE FROM ville WHERE nom = 'Dakar' ;
  6          IF (SQL%NOTFOUND) THEN
  7              RAISE exception_aucune_suppression ;
  8          END IF;

  9      EXCEPTION          /* Section OBLIGATOIRE si instruction RAISE */
 10          WHEN exception_aucune_suppression THEN
 11              DBMS_OUTPUT.PUT_LINE('Aucune suppression !');
 12      END;
 13  /
```

PL/SQL - Exceptions

Exceptions internes NON prédéfinies

- Exception Oracle ayant un numéro, mais pas de nom
- Directive PRAGMA EXCEPTION_INIT adressée au compilateur
 - Associer un nom d'exception à un code d'erreur Oracle existant

```

SQL> DECLARE
      erreur_il_reste_une_ville EXCEPTION;
      PRAGMA EXCEPTION_INIT(erreur_il_reste_une_ville , -2292);
BEGIN
    DELETE FROM pays WHERE nom = 'Senegal';
    DBMS_OUTPUT.PUT_LINE('Pays Sénégal détruit !');
EXCEPTION
    WHEN erreur_il_reste_une_ville THEN
        DBMS_OUTPUT.PUT_LINE('Il reste une ville pour le Sénégal');
    WHEN OTHER THEN
        DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle ' || SQLERRM ||
                             '(' || SQLCODE || ')');
END;
-- # ORA-02292: violated integrity constraint (...)-
      child record found #

```

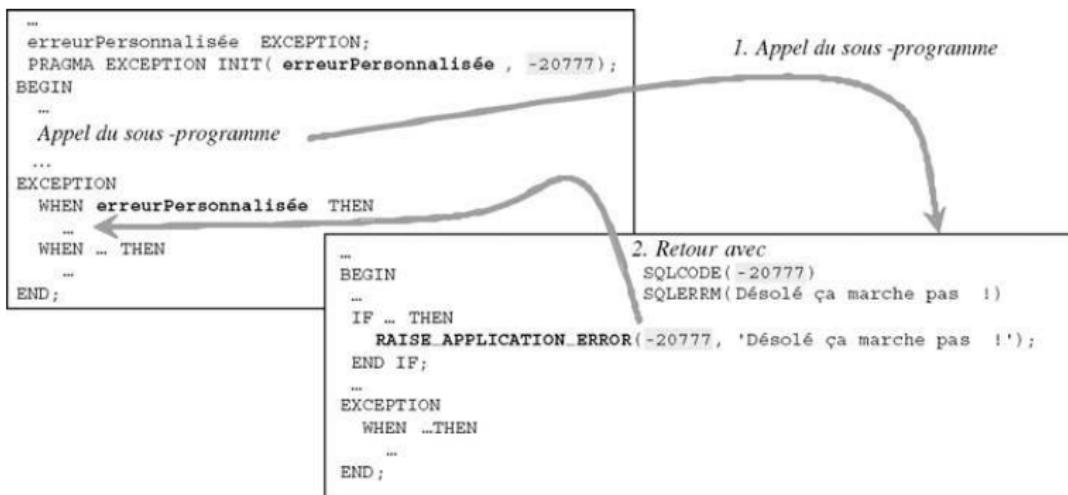
PL/SQL - Exceptions

Exceptions **utilisateur** - Association d'un code d'erreur

- Directive RAISE_APPLICATION_ERROR: définir ses propres messages et codes d'erreurs

RAISE_APPLICATION_ERROR(numeroErreur, message)

- numeroErreur** : valeur comprise entre –20000 et –20999 ;



PL/SQL - Blocs nommés : Procédures et Fonctions

- Blocs PL/SQL **nommés**. Simple changement de déclaration
- **Catalogage**: blocs compilés et stockés dans la base
 - Objets visibles dans le dictionnaire : {ALL | USER}_PROCEDURES
GRANT EXECUTE ON nom_procedure TO utilisateur ;
- **Compilation**: recompilation automatique lors d'une invocation si un objet cité dans le code (table, vue, champ...) a été modifié.
- **Invocation**: dans tout utilitaire Oracle (SQL*Plus, SQL*Forms ...) ou dans un programme externe (C, JAVA, ...) reconnu par la base
- **Imbrication**: sous-fonctions et sous-procédures.
- En **PL/SQL**, mais on peut définir des fonctions ou procédures C ou JAVA utilisables dans des programmes PL/SQL (*external procedures*)
- **Paquetages**
 - Notion classique de *package*: regroupement d'objets PL/SQL (procédures, fonctions, exceptions ...) formant un ensemble cohérent.

PL/SQL - Blocs nommés : Procédures et Fonctions

Structure générale du code

- Directive de définition: CREATE FUNCTION / PROCEDURE
- Déclaration des variables puis déclaration et définition des sous-prog
- Corps du programme
- Gestion des exceptions (erreurs)

```
CREATE PROCEDURE nom_proc ...
IS
-- declarations variables
-- declarations ss_prog
BEGIN
-- code
[EXCEPTION
-- code gestion erreurs]
END nom_proc;
```

```
PROCEDURE ss_prog
IS
-- declarations
BEGIN
-- code
EXCEPTION
-- code gestion erreurs
END ss_prog;
```

PL/SQL - Blocs nommés : Procédures et Fonctions

Procédure

```
CREATE [OR REPLACE] PROCEDURE [schema].nom_proc [( param1 [IN | OUT |
    IN OUT] typeSQL [, param2 ...] )]
    [AUTHID { CURRENT_USER | DEFINER }]
{IS | AS}
    -- declarations variables / ss_prog
BEGIN
    -- code
END [ nom_proc ] ;
```

Fonction

```
CREATE [OR REPLACE] PROCEDURE [schema].nom_proc [( param1 [IN | OUT |
    IN OUT] typeSQL [, param2 ...] )]
    [AUTHID { CURRENT_USER | DEFINER }]
RETURN typeSQL {IS | AS}
    -- declarations variables / ss_prog
BEGIN
    -- code
END [ nom_proc ] ;
```

PL/SQL - Blocs nommés : Procédures et Fonctions

Recompilation manuelle

- `ALTER {FUNCTION | PROCEDURE} nomObjet COMPILE ;`

Destruction

- `DROP {FUNCTION | PROCEDURE} [schéma.]nomObjet ;`

Sous-programmes

- Toute procédure ou fonction peut-être utilisée dans un programme.
 - Droit d'exécution nécessaire.
- On peut également définir un sous-programme imbriqué (*nested subprogram*) dans un programme donné.
 - Le sous-programme n'existera que lors de l'exécution du programme.
 - Déclaration à faire impérativement après les variables du programme principal.
 - Même syntaxe que pour un programme, le CREATE en moins.

PL/SQL - Blocs nommés : Procédures et Fonctions

Fonction factorielle

```
CREATE OR REPLACE FUNCTION factorielle (n IN NUMBER) RETURN NUMBER
IS
BEGIN
    IF n <= 1 THEN
        RETURN 1;
    ELSE
        RETURN n * factorielle (n - 1);
    END IF;
END;
/
-- <== Directive de compilation
```

Procédure factorielle

```
CREATE OR REPLACE FUNCTION facto_proc (n IN NUMBER) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('La factorielle de ' || n || ' est: ' ||
                         factorielle (n));
END;
/
-- <== Directive de compilation
```

PL/SQL - Blocs nommés : Procédures et Fonctions

Compilation

- Erreurs de compilation: détaillées dans le dictionnaire de données:
 - USER_ERRORS, ALL_ERRORS
- Directive SQL simplifiée: SHOW ERRORS

```
SQL> CREATE OR REPLACE PROCEDURE testErreur (n IN NUMBER) IS
      BEGIN
          n := 15 ;
      END;
/
Warning: Procedure created with compilation errors.
```

```
SQL> show errors
Errors for PROCEDURE TESTERREUR:
```

LINE/COL	ERROR
4/3	PL/SQL: Statement ignored
4/3	PLS-00363: expression 'N' cannot be used as an assignment target

PL/SQL - Blocs nommés : Procédures et Fonctions

Exécution

- Sous SQL*Plus (EXECUTE). Dans un bloc PL/SQL (appel direct)

```
SQL> VARIABLE n NUMBER  
SQL> EXECUTE :n := factorielle (5) ;
```

```
PL/SQL procedure successfully completed.
```

```
SQL> PRINT n
```

N

120

```
/* # OU # SQL> SELECT factorielle (:n) FROM DUAL; */
```

```
SQL> VARIABLE n NUMBER;  
SQL> BEGIN  
2      :n := 10 ;  
3      facto_proc (:n) ;  
4  END;  
5/  
La factorielle de 10 est: 3628800  
PL/SQL procedure successfully completed.
```

PL/SQL - Paquetages (*packages*)

Paquetages

- Regroupe différents types d'éléments: types de données (RECORD...), variables, procédures, fonctions, curseurs, exceptions, ...

Spécification

- Ensemble des signatures des éléments empaquetés

```
CREATE [OR REPLACE] PACKAGE nom_package
AS | IS
    -- declarations données / types / ss_prog
END [nom_package];
```

Implémentation:

- Totalité des définitions des éléments du paquetage

```
CREATE [OR REPLACE] PACKAGE BODY nom_package
AS | IS
    -- définitions données / types /
    -- corps des procedure / fonctions
END [nom_package];
```

Exemple d'appel: SELECT nom_pack.factorielle(5) FROM dual ;

PL/SQL - Triggers - Déclencheurs

Triggers / Déclencheurs

- Programme se déclenchant automatiquement suite à un événement ou au passage à un état donné de la base de données.
- Règles de gestion non modélisables par des contraintes d'intégrité
- Sécurité et Audit. Porte sur des **tables** ou des **vues**
- Trois parties : (a) description de l'événement traqué, (b) conditions éventuelles de déclenchement sur l'événement, (c) action à réaliser.

Différents types de déclencheurs

- Sur ordre LMD (INSERT, ...). Sur ordre LDD (CREATE, ALTER...)
- Sur événement Oracle (démarrage, arrêt ...): déclencheur d'instance

Déclencheur LMD

- Sur l'**état** global de l'objet considéré : déclencheur d'état
- Sur **chaque tuple** modifié : déclencheur de ligne

Triggers - Déclencheurs

Création: Syntaxe simplifiée

```

CREATE [ OR REPLACE ] TRIGGER nom_trigger
BEFORE | AFTER | INSTEAD OF      -- INSTEAD OF ==> Vue multitable
INSERT | DELETE | UPDATE [ OF liste_colonne ]
ON {[schéma.]nom_table | nom_vue}
[FOR EACH ROW]
[WHEN condition]
BEGIN
    code_trigger                  -- Bloc PL/SQL
END;

```

Activation / Désactivation - Suppression

```

ALTER TRIGGER nom_trigger ENABLE | DISABLE
DROP TRIGGER nom_trigger

```

À noter ...

- Instructions interdites: COMMIT, ROLLBACK, SAVEPOINT ...
- Attention à ne pas créer de déclencheurs récursifs

PL/SQL - Triggers - Déclencheurs

Déclencheurs LMD

Nature de l'événement	État (<i>statement trigger</i>) sans FOR EACH ROW	Ligne (<i>row trigger</i>) avec FOR EACH ROW
BEFORE	Exécution une fois avant la mise à jour.	Exécution avant chaque ligne mise à jour.
AFTER	Exécution une fois après la mise à jour.	Exécution après chaque ligne mise à jour.

Directives :NEW et :OLD

- Nouvelle et ancienne valeur: uniquement pour déclencheurs de ligne

Nature de l'événement	:OLD.colonne	:NEW.colonne
INSERT	NULL	Nouvelle valeur.
UPDATE	Ancienne valeur.	Nouvelle valeur.
DELETE	Ancienne valeur.	NULL

PL/SQL - Triggers - Déclencheurs

Exemple - (1/4) - Les noms de villes doivent être majuscule !

```
SQL> CREATE OR REPLACE nom_ville_maj
      BEFORE INSERT ON VILLE
      FOR EACH ROW
      BEGIN
          :NEW.nom := UPPER (:NEW.nom) ;
      END;
      /
```

Exemple - (2/4) - Et si quelqu'un fait une mise à jour ??

```
SQL> CREATE OR REPLACE nom_ville_maj
      BEFORE INSERT OR UPDATE /* OF nom */ ON VILLE
      FOR EACH ROW
      BEGIN
          IF (INSERTING /* OR UPDATING */) THEN
              :NEW.nom := UPPER (:NEW.nom) ;
          ELSIF (UPDATING /*('nom')*/ ) THEN
              :NEW.nom := UPPER (:NEW.nom) ;
          END IF;
      END;
```

PL/SQL - Triggers - Déclencheurs

Exemple - (3/4) - Que fait ce trigger ENIGME_1 ?

```
SQL> CREATE OR REPLACE TRIGGER enigme_1
  BEFORE UPDATE ON ville
  FOR EACH ROW
  WHEN (OLD.nom IS NOT NULL OR OLD.population IS NOT NULL)
  BEGIN
    IF (:OLD.nom IS NOT NULL) THEN
      :NEW.nom := :OLD.nom ;
    END IF;
    IF (:OLD.population IS NOT NULL) THEN
      :NEW.population := :OLD.population ;
    END IF;
    IF (:NEW.num_v != :OLD.num_v) THEN
      RAISE_APPLICATION_ERROR (-20102, 'Plus de doute ? ') ;
    END IF;
    -- et si OLD.nom = OLD.population = NULL ?
```

- Expression SQL comme condition dans la clause WHEN.
- Les pseudo enregistrements OLD et NEW s'écrivent sans le symbole ":"

PL/SQL - Triggers - Déclencheurs

Exemple - (4/4) - Que fait ce trigger ENIGME_2 ?

```
SQL> CREATE OR REPLACE TRIGGER enigme_2
  AFTER INSERT OR UPDATE OR DELETE ON ville
  FOR EACH ROW
  DECLARE
    nom_user VARCHAR2(30);
    action    VARCHAR2(6);
  BEGIN
    SELECT user INTO nom_user FROM dual;

    IF (INSERTING) THEN      action := 'INSERT'  ;
    END IF;
    IF (UPDATING) THEN      action := 'UPDATE'  ;
    END IF;
    IF (DELETING) THEN      action := 'DELETE'  ;
    END IF;

    INSERT INTO table_audit
      VALUES (maseq.NEXTVAL, nom_user, action) ;

  END;
```

PL/SQL - Triggers - Déclencheurs

Déclencheur d'état LMD (*statement triggers*)

- Déclaré **sans** la directive FOR EACH ROW
- Porte sur la globalité de la table et non sur chaque enregistrement.

```
CREATE TRIGGER REPOS
BEFORE DELETE OR UPDATE INSERT ON ville

BEGIN
    IF TRIM (TO_CHAR (SYSDATE, 'DAY')) IN ('SATURDAY', 'SUNDAY') THEN
        RAISE_APPLICATION_ERROR (-20102, 'Désolé, pas de M&J le W.E. );
    END IF;
END;
/

-- TO_CHAR (SYSDATE, 'DAY'): renvoie la date (jour) sous forme
-- de chaîne de 9 caractères.
-- TRIM: suppression des caractères blancs en début et fin de chaîne
```

PL/SQL - Triggers - Déclencheurs

Déclencheur INSTEAD OF

- Uniquement sur des **vues**:
- Mise à jour d'une vue multitable qui ne pouvait être modifiée directement par INSERT, UPDATE ou DELETE
- Le déclencheur fera des actions **au lieu** d'insérer, de modifier ...
- Peut aussi être utilisé pour les BD réparties par liens (*database links*).
- Fait intervenir la clause FOR EACH ROW
- Ne fait pas intervenir les options BEFORE et AFTER.
- L'option WITH CHECK OPTION d'une vue n'est pas vérifiée
- Il n'est pas possible de spécifier une liste de colonnes pour INSTEAD OF UPDATE. Le déclencheur s'exécutera quelle que soit la colonne modifiée.
- Il n'est pas possible d'utiliser la clause WHEN

PL/SQL - Déclencheurs INSTEAD OF

Exemple de sessions - (1/3) - État de l'instance

```
SQL> select * from ville;
NUM_V    NOM                  POP
-----  -----
1        Dakar                50000
3        Saint Louis          666
6        Louga               35000
```

```
SQL> select * from visite;
NUM_V    NUM_P
-----  -----
1        1
1        3
3        3
```

```
SQL> select * from ville_visitee
NOM_PERS      NOMVILLE
-----  -----
DIOP          Dakar
MBENGUE       Dakar
MBENGUE       Saint Louis
```

```
SQL> select * from personne;
NUM_P    NOM
-----  -----
1        DIOP
2        FALL
3        MBENGUE
```

```
SQL> create or replace view ville_visitee
as select p.nom as nom_pers,
v.nom as nom_ville from personne p,
ville v, visite vi where
p.num_p = vi.num_p and
vi.num_v = v.num_v with check option;
```

```
SQL> insert into ville_visitee
values ('LY', 'Louga');
ERROR at line 1: ORA-01779: cannot
modify a column which maps to a non
key-preserved table
```

PL/SQL - Déclencheurs INSTEAD OF

Exemple de sessions - (2/3) - Crédation des objets

-- Crédation d'une séquence pour générer les clés de ville et personne

```
CREATE SEQUENCE maseq START WITH 100 INCREMENT BY 1;
```

-- Crédation du trigger instead of

```
CREATE OR REPLACE TRIGGER insere_dans_vue_ville_visitee  
INSTEAD OF INSERT ON ville_visitee
```

```
FOR EACH ROW
```

```
DECLARE
```

```
    new_id NUMBER ; -- Nouvel ID (chaque nouveau tuple (pers et ville))
```

```
BEGIN
```

```
    SELECT maseq.NEXTVAL INTO new_id FROM dual ;
```

```
    INSERT INTO ville VALUES (new_id, :new.nom_ville, null) ;
```

```
    INSERT INTO personne VALUES (new_id, :new.nom_pers) ;
```

```
    INSERT INTO visite VALUES (new_id, new_id);
```

```
END;
```

PL/SQL - Déclencheurs INSTEAD OF

Exemple de sessions - (3/3) - Nouvel état de l'instance

```
SQL> insert into ville_visitee values ('Ly', 'Louga') ;
```

```
SQL> select * from ville;
```

NUM_V	NOM	POP
1	Dakar	50000
3	Saint Louis	666
6	Louga	35000
100	Louga	

```
SQL> select * from visite;
```

NUM_V	NUM_P
1	1
1	3
3	3
100	100

```
SQL> select * from personne;
```

NUM_P	NOM
1	DIOP
2	FALL
3	MBENGUE
100	Ly

```
SQL> select * from ville_visitee
```

NOM_PERS	NOM_VILLE
DIOP	Dakar
MBENGUE	Dakar
MBENGUE	Saint Louis
Ly	Louga

PL/SQL - Déclencheurs LDD & d'Instance

Déclencheur **LDD**: modification de la structure de BD

```
CREATE [OR REPLACE] TRIGGER [schéma.]nom_trigger  
{ BEFORE | AFTER } action_LDD  
ON {[schéma.]SCHEMA| DATABASE }  
Bloc PL/SQL ....
```

- Actions LDD : CREATE, ALTER, DROP, COMMENT, GRANT, etc...

Déclencheur **d'Instance**: événement Oracle

```
CREATE [OR REPLACE] TRIGGER [schéma.]nom_trigger  
{ BEFORE | AFTER } event_Oracle  
ON {[schéma.]SCHEMA| DATABASE }  
Bloc PL/SQL ...
```

- Événements Oracle:
 - Serveur Oracle: STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR ...
 - Erreur spécifique SGBD: NO_DATA_FOUND, DUP_VAL_ON_INDEX ...

BDA Oracle

FIN

- C. Soutou (2008): SQL pour Oracle, Applications avec Java, PHP et XML, 3^{ème} édition, Eyrolles
- J. Gennick, S. Mishra: Oracle SQL*Loader - The Definitive Guide, O'REILLY, 2001
- Supports Pr. Jean-Yves Antoine
(<http://www.info.univ-tours.fr/~antoine>)
- Etc.