

# Cours ORACLE

2019

# Présentation SQL

- Fondement SQL (*Structured Query Language*) : milieu 70
- Crée par IBM et commercialisé par Oracle
- Normalisation :
  - l'ANSI (American National Standards Institute) dans les documents ANSI 90751:1999, ANSI 90752:1999 et ANSI 90755:1999.
  - l'ISO (International Organisation for Standardization) dans les documents ISO/IEC 90751:1999, ISO/IEC 90752:1999 et ISO/IEC 90755:1999.

# Présentation SQL

- PL/SQL : Utilisé tant du côté serveur que du côté client.
- Syntaxe générale ressemble à celle des langages Pascal et Ada.
- Permet de combiner des requêtes SQL et des instructions procédurales (boucles, conditions...), dans le but de créer des traitements complexes.
- PL/SQL : Car SQL est non procédural.

# Présentation SQL - ORACLE

1979 : Oracle 2. 1<sup>ère</sup> version commerciale. Premier SGBD basé sur le SQL de CODD.

1983 : Oracle 3. Réécrit en C.

1984 : Oracle 4. Gestion des transactions.

1992 : Oracle 7. Contraintes référentielles. Procédures stockées. Triggers.

1997 : Oracle 8. Objet-relationnel.

1998 : Oracle 8i. i pour internet.

2004 : Oracle 10g. g pour grid computing : calcul distribué et gestion de cluster.

2005 : Oracle 10g express édition. Version gratuite de Oracle 10g mais bridée en nombre de processeurs, d'enregistrements (4G0) et de mémoire (1G0).

2007 : Oracle 11g. Amélioration (performance et de facilité d'administration).

2009 : Oracle achète Sun (Java) qui avait acheté MySQL en 2008.

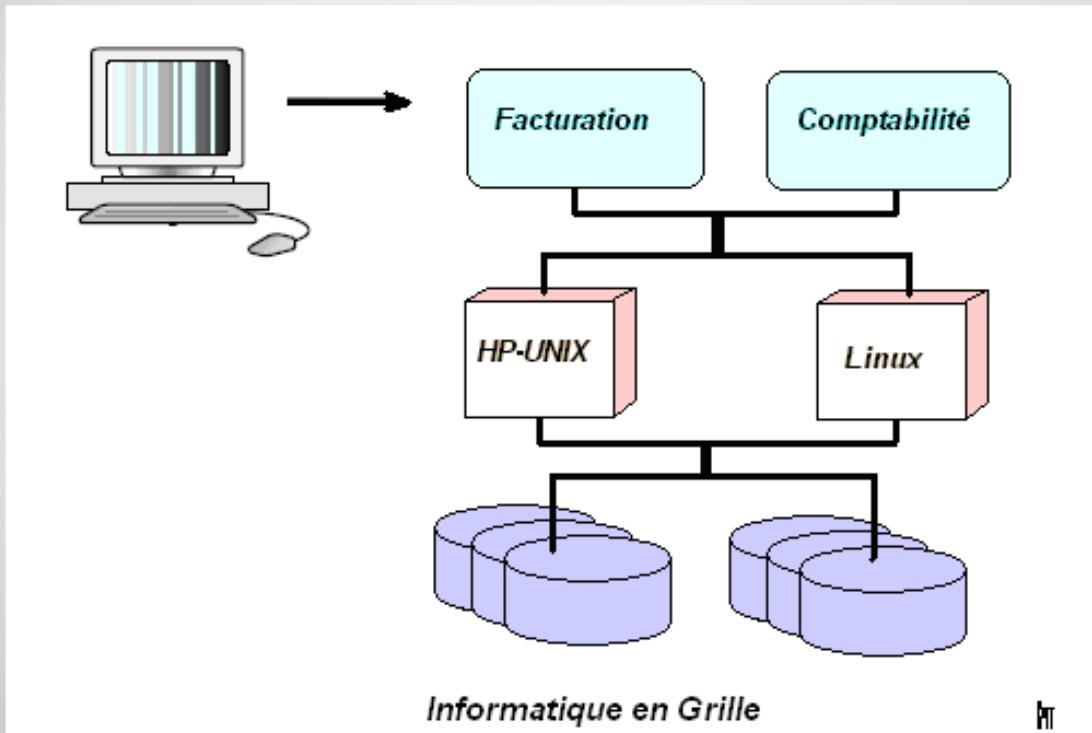
2013 : Oracle 12c (12,1 : release 1, etc).

2018 : Oracle 18c (2 processeurs, 2 Gb de mémoire vive et 12 Gb de données)

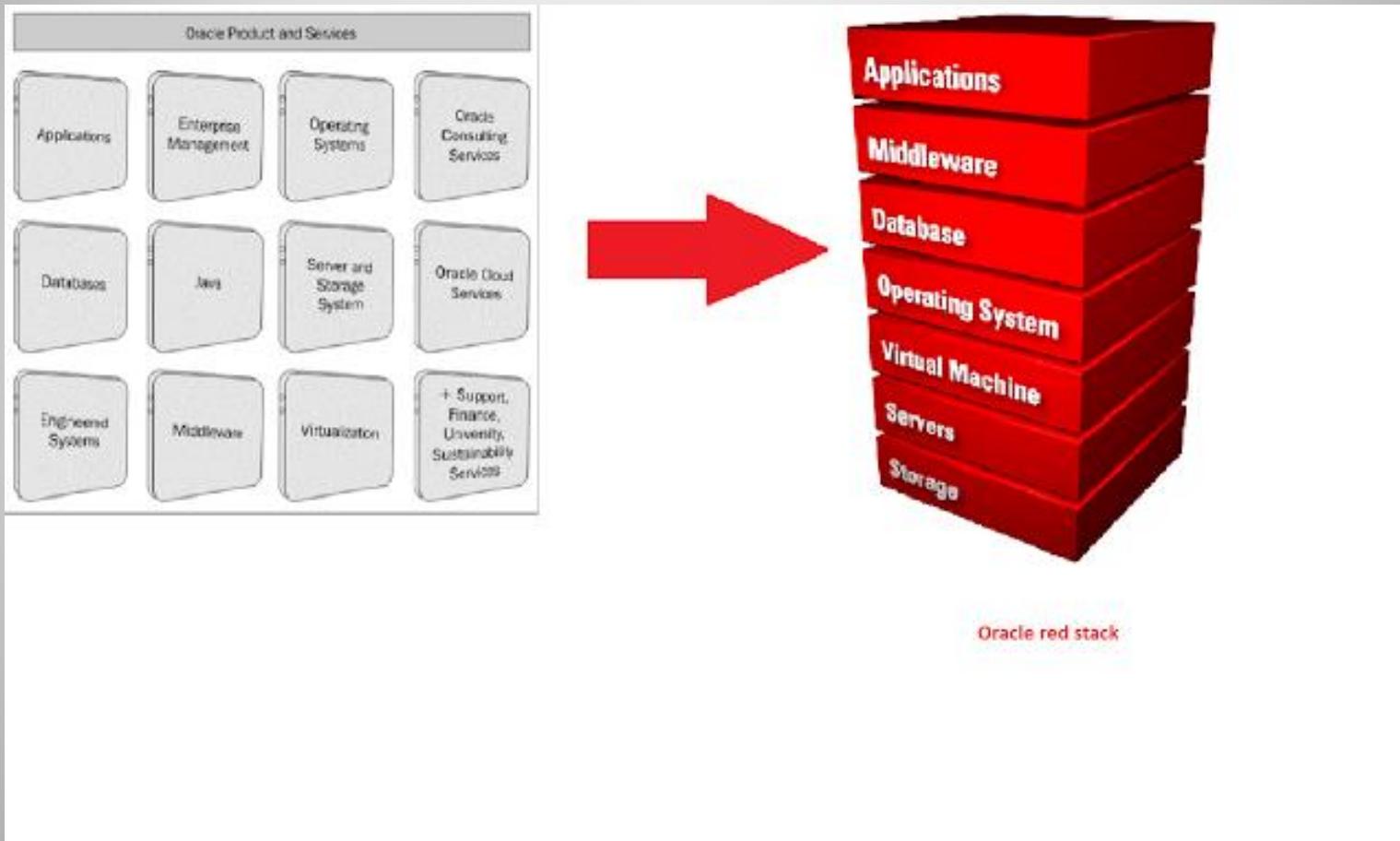
# Présentation SQL - ORACLE

- Actuellement : version 18c release 2 / 12c
- Serveur HTTP intégré à la base de donnée
  - utilise la technologie WebDAV
  - Implémenté sous le nom de XML DB
  - Nommé par Oracle « Embedded PL/SQL Gateway ».
- Grid computing : permet de constituer des matrices de serveurs et de systèmes de stockage économiques (pools de ressources).

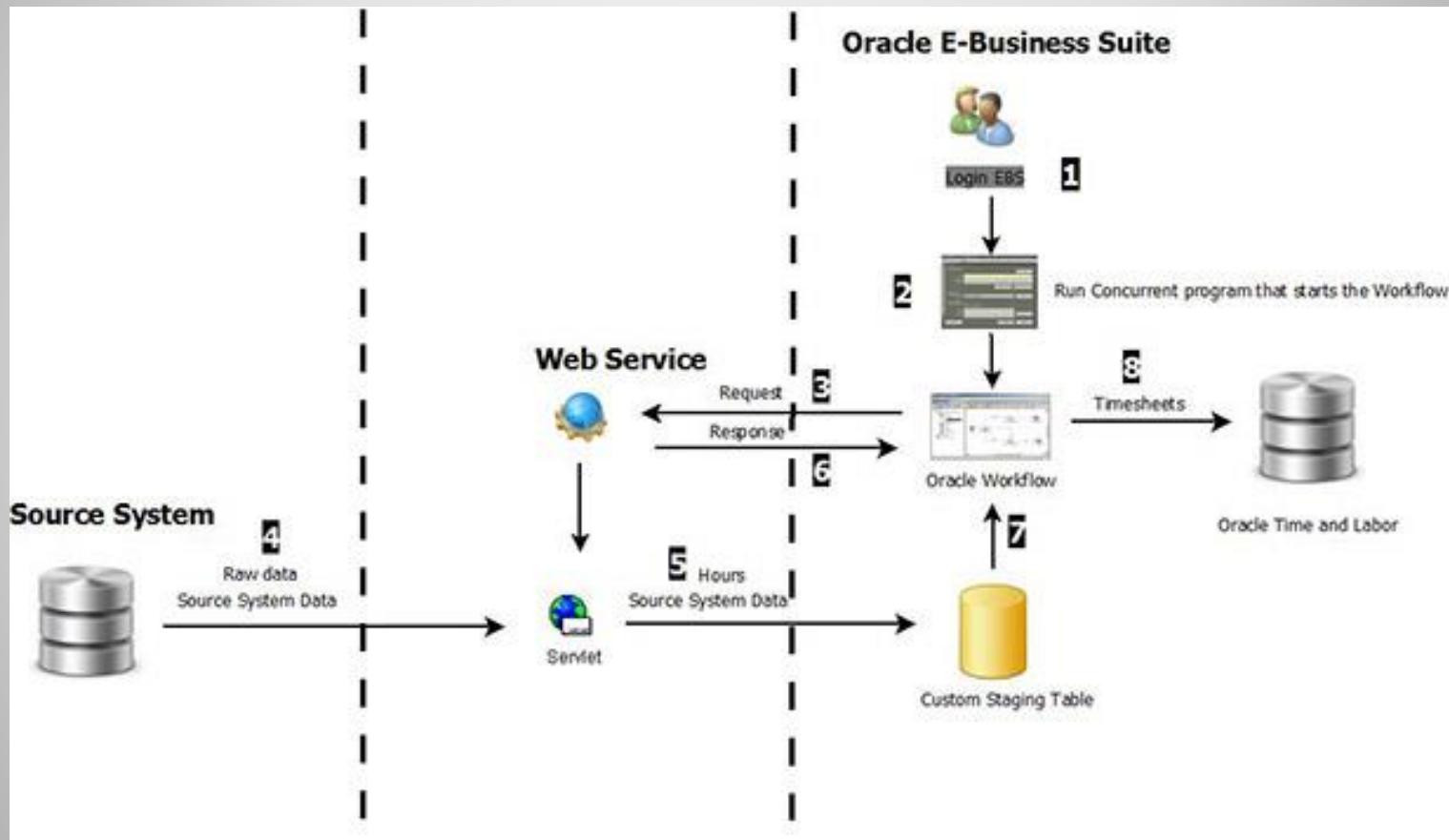
# ORACLE – Exemple de Grid



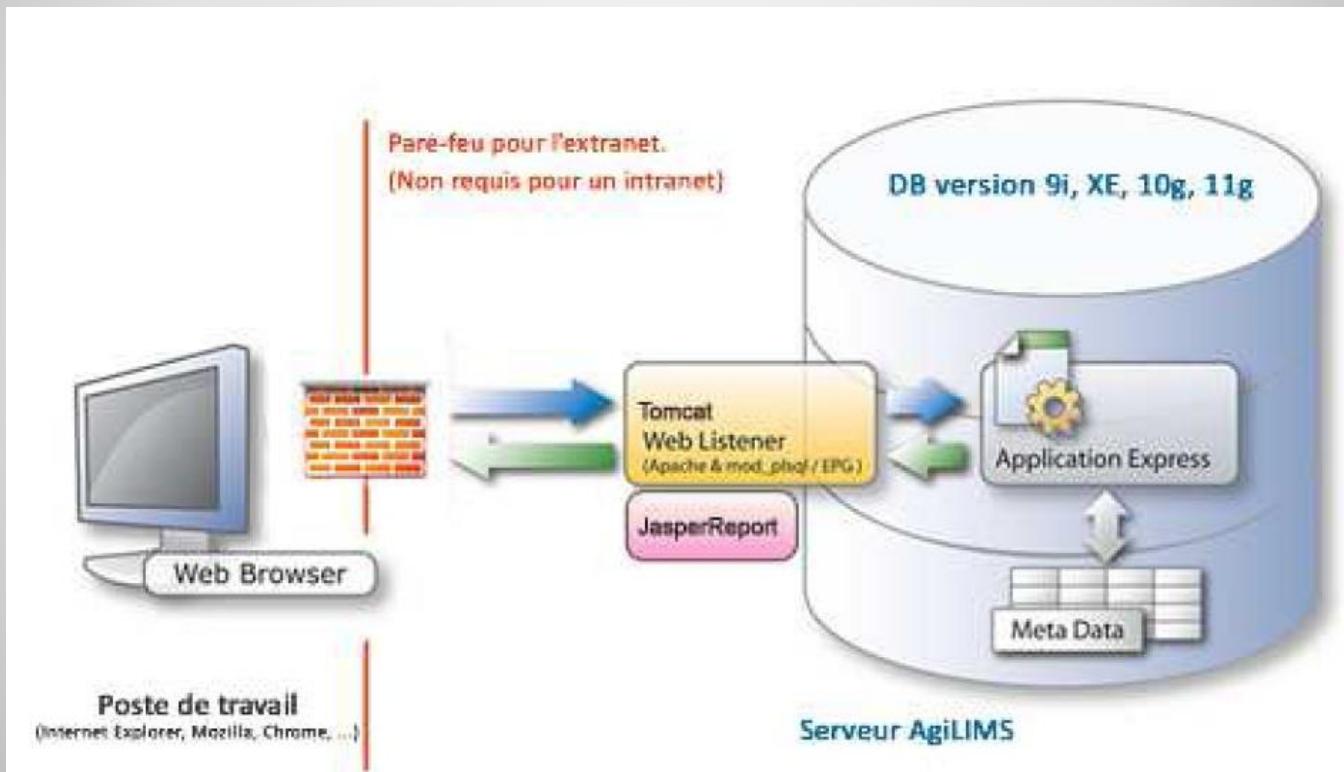
# Présentation Oracle (1/3)



# Présentation Oracle (2/3)



# Présentation Oracle (3/3)



# ORACLE - Définitions

- Cardinalité d'un domaine
- Produit cartésien
- Intégrité référentielle
- Clé étrangère
- Notion de schéma



# ORACLE - SCHEMA

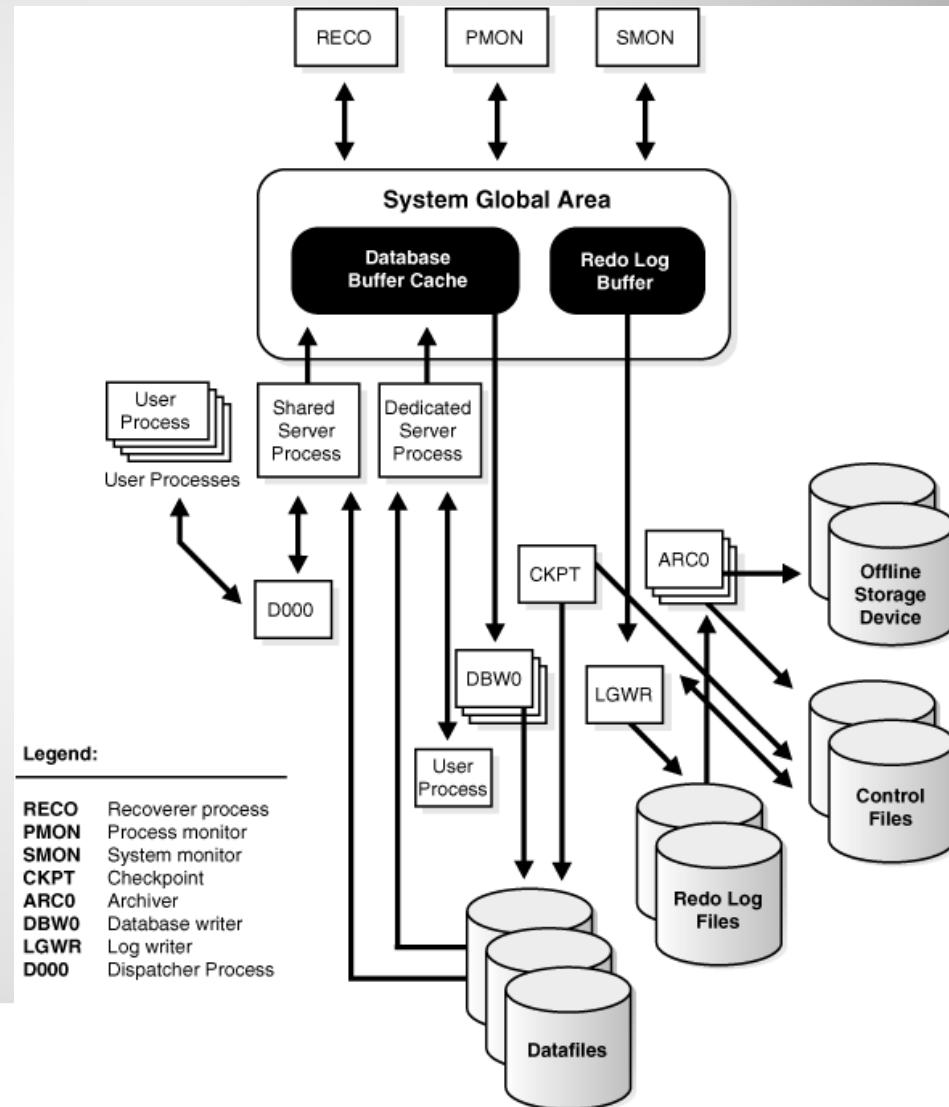
Schéma :

- Ensemble des objets qui appartiennent à un utilisateur, ces objets sont préfixés par le nom de l'utilisateur qui les a créés.
- Notion logique désignant la totalité des objets créés par un utilisateur.
- Table BETTY.AVION différente de CHARLY.AVION
- Principaux types d'objets de schéma :
  - Tables et index
  - Directory
  - Vues, séquences et synonymes
  - Programmes PL/SQL (procédures, fonctions, packages, triggers)

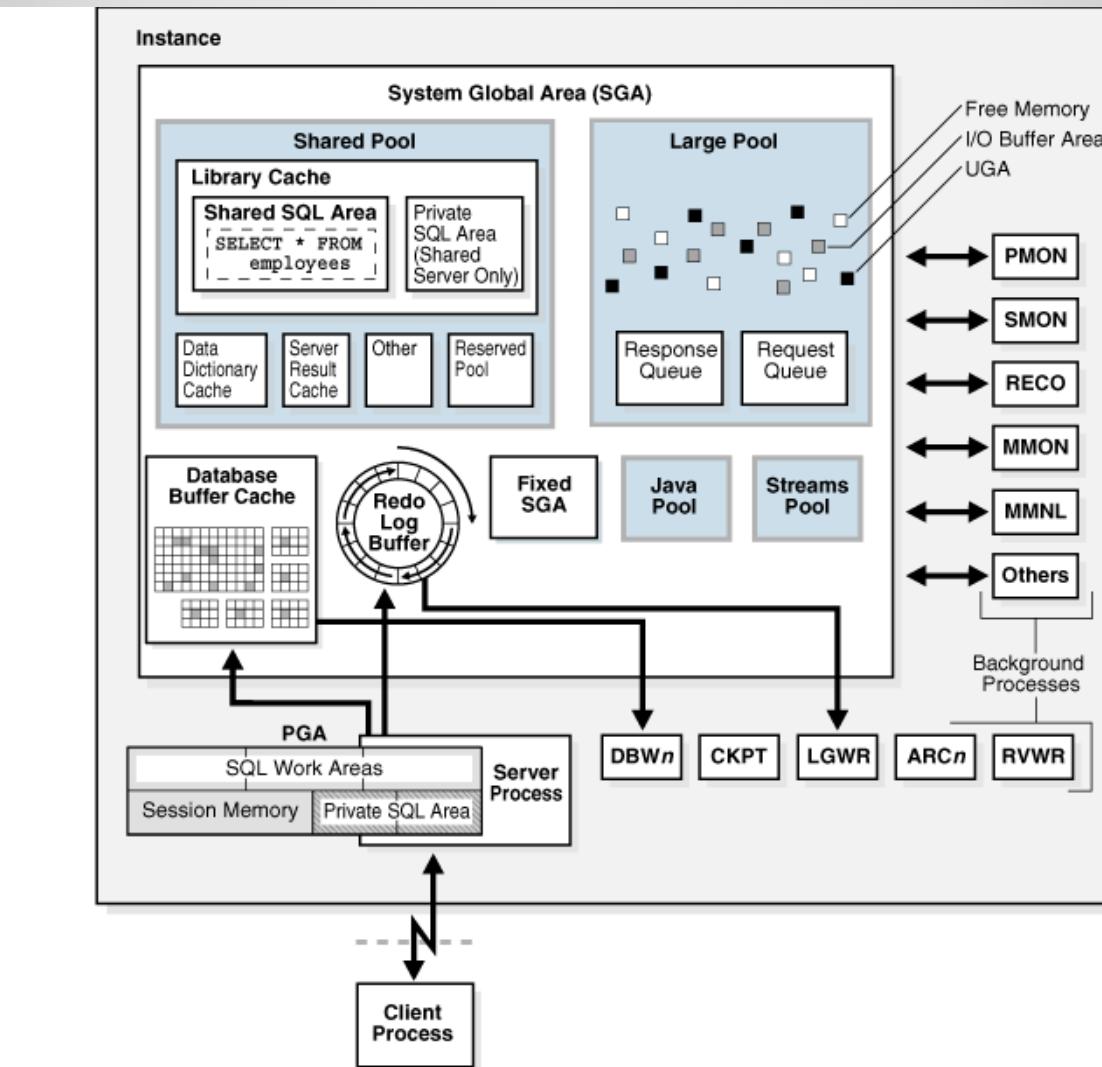
# ORACLE – Architecture Oracle

5 process obligatoires :

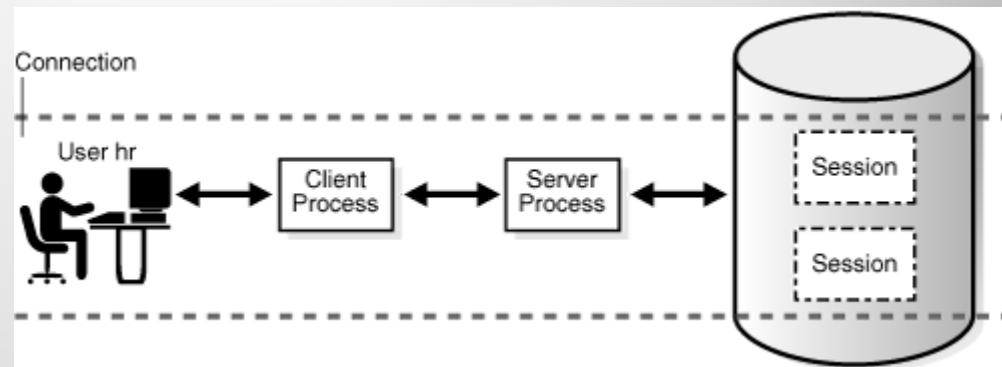
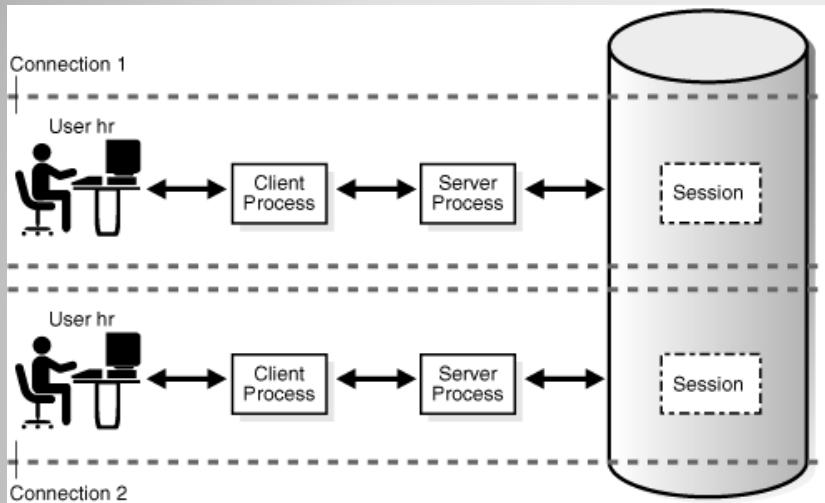
SMON  
PMON  
DWBN  
LGWR  
CKPT



# ORACLE – Architecture Oracle



# ORACLE – Connection / session



# Etapes de résolution d'un problème (1/3)

- Analyser le besoin
  - Transcrire sous forme de relation résultante des besoins exprimés par le prescripteur.
  - Déterminer les attributs et les relations à utiliser.
  - Exprimer les calculs élémentaires et d'agrégats pour créer les attributs inexistant
- Établir la "vue"

# Etapes de résolution d'un problème (2/3)

- ① Relations concernées.
- ② Restrictions (pour éliminer les lignes inutiles).
- ③ Jointures, Produits cartésiens, Unions, Intersections, Différences (pour associer les lignes restantes).
- ④ Calculs élémentaires (pour créer les nouvelles colonnes).
- ⑤ Calculs d'agrégats (pour les colonnes statistiques).

## Etapes de résolution d'un problème (3/3)

- ⑥ Jointure entre la table obtenue en ⑤ et la table initiale en ④ (pour ajouter les colonnes statistiques aux autres).
- ⑦ Répéter les étapes du ⑤ pour les autres regroupements.
- ⑧ Restrictions par rapport aux attributs calculés.
- ⑨ Projections pour éliminer les doublons.
- ⑩ Projection finale pour éliminer les attributs inutiles dans la table résultante.

# Objets d'une base de donnée (1/3)

- Table
- Index
- View
- Synonym
- Sequence
- Snapshots
- Database links

# Objets d'une base de donnée (2/3)

Stockage physique :

- Cluster
- Tablespace
- Directory

# Objets d'une base de donnée (3/3)

Stockage d'instructions :

- Schema
- Procedure
- Function
- Database trigger
- Package
- Library

# Dictionnaire de données (1/2)

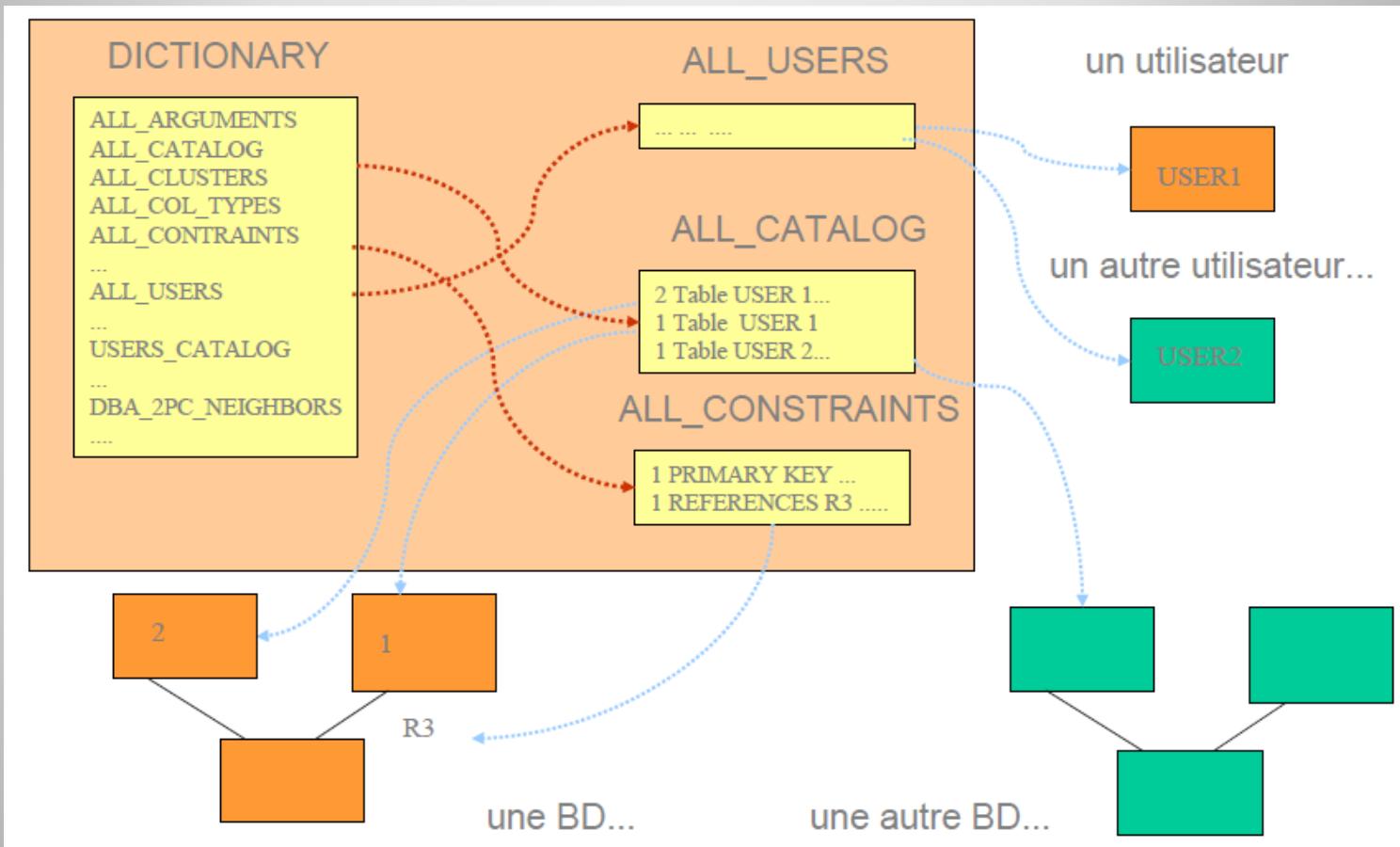
Ensemble de tables et de vues : informations sur le contenu d'une base de données.

- Les structures de stockage.
- Les utilisateurs et leurs droits.
- Les objets (tables, vues, index, procédures, ...).

Il appartient à l'utilisateur SYS et est stocké dans le tablespace SYSTEM. Sauf exception, toutes les informations sont stockées en MAJUSCULE.

Est chargé en mémoire et utilisé par Oracle pour traiter les commandes SQL.

# Dictionnaire de données (2/2)



# Les tables et vues statiques

Basées sur de vraies tables stockées dans le tablespace SYSTEM, et sont accessibles uniquement quand la base est ouverte.

- USER\_\* : Informations sur les objets qui appartiennent à l'utilisateur
- ALL\_\* : Information sur les objets auxquels l'utilisateur a accès (les siens et ceux sur lesquels il a reçu des droits)
- DBA\_\* : Information sur tous les objets de la base

# Accès au dictionnaire de donnée

Les vues DICTIONARY et DICT\_COLUMNS donnent la description de toutes les tables et vues du dictionnaire (statiques et dynamiques).

la liste complète des vues statiques est obtenue par la requête :

```
SELECT view_name FROM ALL_VIEWS  
WHERE ALL_VIEWS like 'DBA*_%' escape '*' ;
```

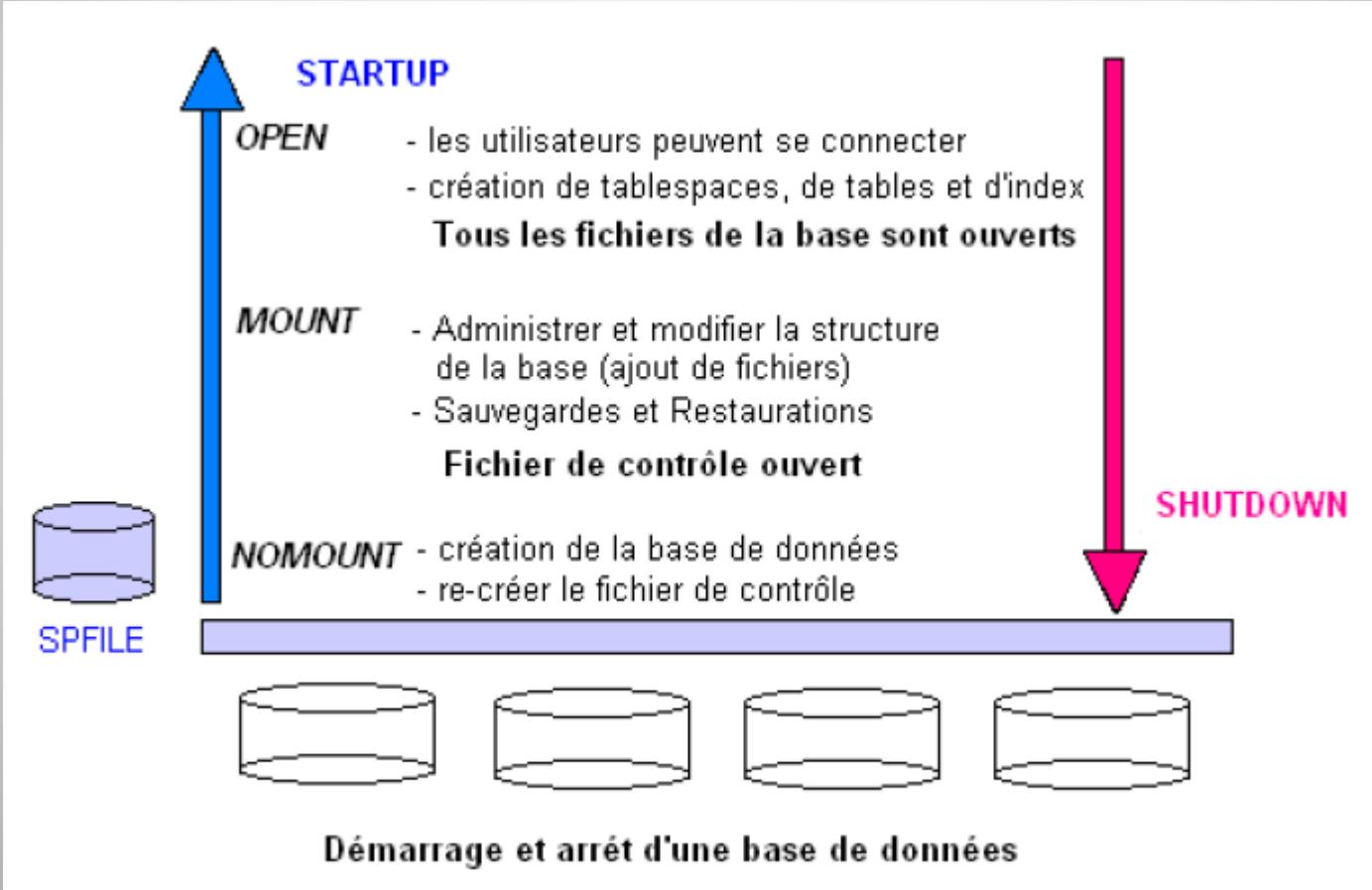
# Les tables et vues dynamiques de performance

- Basées sur des informations en mémoire ou extraites du fichier de contrôle.
- Informations sur le fonctionnement de la base de données, notamment sur les performances.
- Remises à zéro si on arrête la base de données.
- Accessibles même lorsque la base n'est pas complètement ouverte (MOUNT).

# Vues dynamiques de performance

- Préfixées par « V\$ »
- Derrière le préfixe, le reste du nom de la vue est représentatif de l'information accessible :
  - V\$instance
  - V\$DATABASE
  - V\$SGA
  - V\$PARAMETER

# Etat d'une base

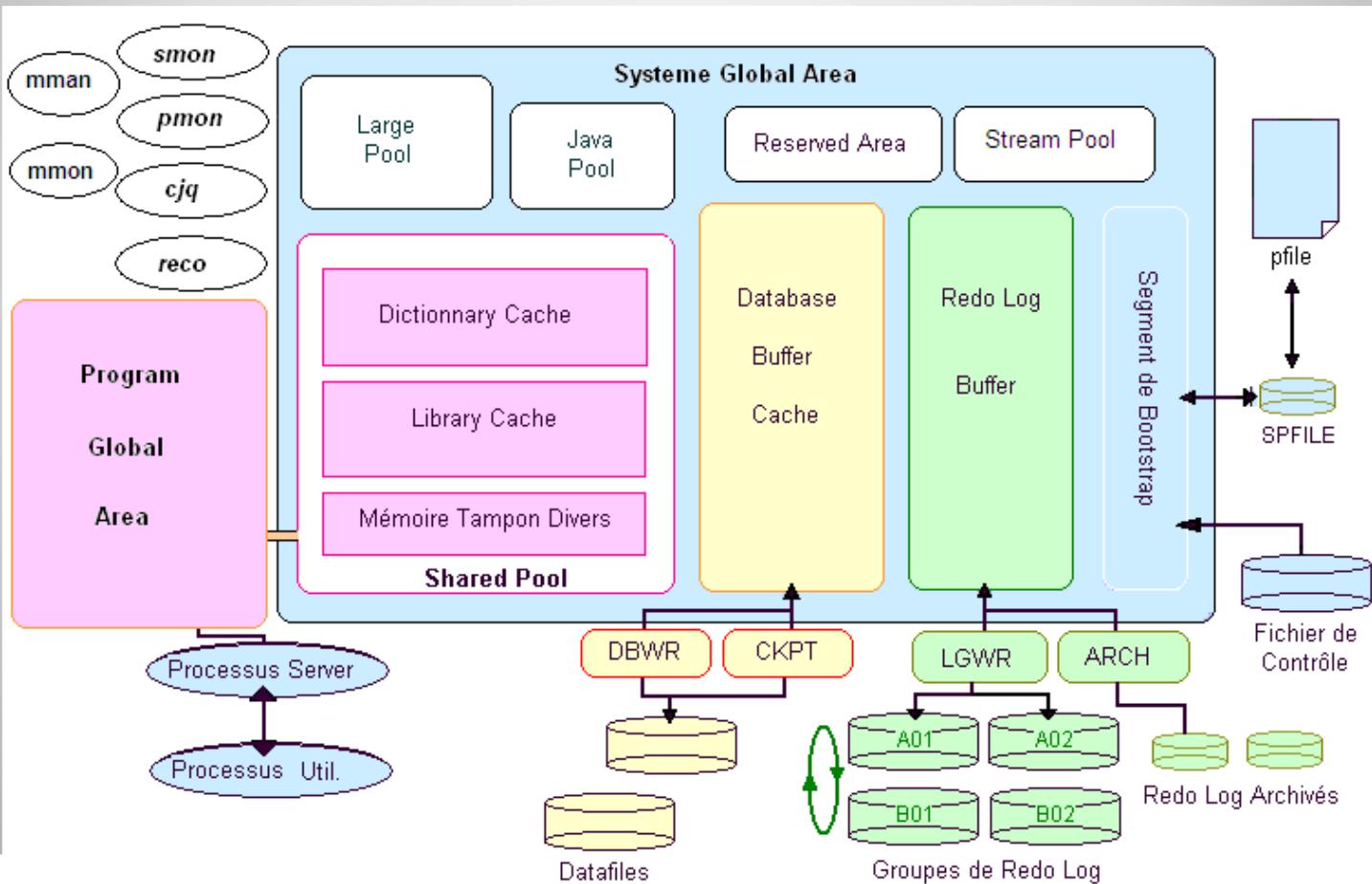


## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Architecture Oracle



# Architecture Oracle (1/2)

Une instance et une base de donnée

Instance :

- Une zone de mémoire partagée appelée System Global Area (SGA).
- Un ensemble de processus d'arrière plan ayant chacun un rôle bien précis.
- Un ensemble de processus serveur chargés de traiter les requêtes des utilisateurs.

# Architecture Oracle (2/2)

Base de donnée :

- Un fichier de contrôle, contenant les informations sur tous les autres fichiers de la base (nom, emplacement, taille).
- Fichiers de Redo Log (activité des sessions connectées à la base) : journaux de transactions de la base. Organisés en groupe possédant le même nombre de membres.
  - Et éventuellement, de fichiers de Redo Log archivés contenant les archives d'anciens fichiers de Redo Log.
- Un ou plusieurs fichiers de données qui contiennent les données des tables de la base.

# Accéder à une base distante

- Le processus d'écoute LISTENER placé sur le serveur (coté instance de base de données) permet la connexion des clients à l'instance.
- Celle-ci est vue comme un service (paramètre SERVICES\_NAME) dans l'instance). Le LISTENER est configuré via le fichier LISTNER.ORA.

Plusieurs méthodes de connexions peuvent être utilisées :

- Locale : le fichier TNSNAMES.ORA est configuré sur le poste client et se charge de la résolution du service Oracle Net.
- Simplifiée, (easy connect naming), permettant une connexion via l'adresse du service à travers le réseau TCP/IP.
- LDAP (directory naming), un annuaire LDAP se charge de la résolution du nom de service. Cette méthode nécessite un produit tiers.

# Accéder à une base distante

A l'adresse indiquée le listener reçoit la demande et connecte le client à l'instance <SID> demandée



Le client se connecte en utilisant le nom du service Oracle Net

**connect USER01/motdepasse@Service**  
(le tnsname.ora associe le nom du service à l'adresse du serveur )

*Fonctionnement de Oracle Net*

# LISTENER.ORA

```
# listener.ora Network Configuration File:  
C:\oracle\product\10.2.0\db_1\network\admin\listener.ora  
# Generated by Oracle configuration tools.  
SID_LIST_LISTENER =  
(SID_LIST =  
(SID_DESC =  
(SID_NAME = PLSExtProc)  
(ORACLE_HOME = C:\oracle\product\10.2.0\db_1)  
(PROGRAM = extproc)  
)  
)  
  
LISTENER =  
(DESCRIPTION_LIST =  
(DESCRIPTION =  
(ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))  
(ADDRESS = (PROTOCOL = TCP)(HOST = TELLORA01)(PORT = 1521))  
)  
)
```

# TNSNAMES.ORA

```
# tnsnames.ora Network Configuration File:  
C:\oracle\product\10.2.0\db_1\network\admin\tnsnames.ora  
# Generated by Oracle configuration tools.  
TAHITI =  
(DESCRIPTION =  
  (ADDRESS = (PROTOCOL = TCP)(HOST = TELLORA01)(PORT = 1521))  
  (CONNECT_DATA =  
    (SERVER = DEDICATED)  
    (SERVICE_NAME = tahiti)  
)  
)  
EXTPROC_CONNECTION_DATA =  
(DESCRIPTION =  
  (ADDRESS_LIST =  
    (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))  
)  
  (CONNECT_DATA =  
    (SID = PLSExtProc)  
    (PRESENTATION = RO))
```

# ORACLE – SQL PLUS (1/4)

- Manipulation + exécution de commandes SQL et de blocs PL/SQL.
- Mise en forme des résultats de requêtes.
- Visualisation des structures des tables, copie de données interbase.
- Commandes et opérations d'entrée/sortie (saisie, affichage, manipulation de variables).

Avant Oracle 11g, l'outil SQL\*Plus décliné sous 3 formes différentes :

- outil ligne de commande (sqlplus.exe) ;
- application graphique Windows (sqlplusw.exe) ;
- application Web (iSQL\*Plus).

Depuis la version 11, seule la version ligne de commande subsiste.

# ORACLE – SQL PLUS (2/4)

Connection avec SQL\*PLUS :

Syntaxe

sqlplus[[S] [nomuser[/motpasse]][@chaîne\_de\_connexion]]

- S : Mode silencieux.

Nomuser : Nom de connexion à la base.

Motpasse : Mot de passe de l'utilisateur.

chaîne\_de\_connexion : Nom du service défini dans le fichier TNSNAMES.ORA à utiliser pour se connecter au serveur Oracle.

Sortie : DISCONNECT / EXIT

# ORACLE – SQL PLUS (3/4)

## Exécution des instructions

- Les commandes SQL et PL/SQL peuvent être saisies sur plusieurs lignes (on marque une fin de ligne par la touche [Entrée]), le caractère de fin de commande étant le point virgule (;).
- Pour améliorer la lisibilité on peut insérer dans la syntaxe autant d'espaces ou de tabulations qu'on le souhaite.
- Il n'y a pas de distinction entre majuscules et minuscules (sauf utilisation de guillemets).

# ORACLE – SQL PLUS (4/4)

## Utilisation de scripts

START,@,@@

- Exécution des commandes contenues dans le fichier.

Syntaxe : STA[RT] nomfic ou @nomfic ou @@nomfic

Par défaut, l'extension .sql est prise pour le fichier. Le nom du fichier doit contenir le chemin complet d'accès au fichier si ce dernier n'est pas dans le répertoire courant.

Par défaut, si seul le nom du fichier est indiqué derrière la commande START ou @ alors le fichier script est recherché dans le répertoire de travail de SQLPLUS.

Lancer un script au travers de la commande @@ : permet de garantir que le fichier sera recherché dans le même répertoire que celui du script qui demande l'exécution.

# ORACLE – Personnalisé SQL PLUS

DESC[RIBE] objet

Description de structure de table, vue, synonyme ou code stocké.

SET parametre valeur

AUTOCOMMIT {OFF/ON/IMMEDIATE/n}

Validation automatique.

PAUSE {OFF/ON/texte}

Activation de la pause en fin de page.

SPACE {1/n}

Nombre de caractères séparant les colonnes à l'affichage.

ECHO {OFF/ON}

Affichage des commandes d'un script à l'exécution.

TERMOOUT {ON/OFF}

Affichage du résultat des commandes d'un script.

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# **LANGUAGE PL/SQL**



# Catégories d'instructions (1/5)

DDL (*Data Definition Language*) :

- Ajout, suppression, modification des lignes.
- Visualisation du contenu des tables.
- Verrouillage des tables.

Instructions : INSERT, UPDATE, DELETE, SELECT, EXPLAIN PLAN,  
LOCK TABLE, MERGE.

## Catégories d'instructions (2/5)

DML (*Data Manipulation Language*) :

- Créer, modifier, supprimer des objets.
- Autoriser ou interdire l'accès aux données.
- Activer, désactiver l'audit.
- Commenter le dictionnaire des données.

Instructions : CREATE, ALTER, DROP, GRANT, REVOKE, AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE, COMMENT, FLASHBACK, PURGE.

## Catégories d'instructions (3/5)

Transaction Control language (TCL) :

- Caractéristiques des transactions
- Validation, annulation des modifications

Instructions : COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION, SET CONSTRAINT.

## Catégories d'instructions (4/5)

Session Control language (gestion d'une session utilisateur) :

- Modification des caractéristiques de session.
- Activation, désactivation des privilèges utilisateurs.

Instructions : ALTER SESSION, SET ROLE.

## Catégories d'instructions (5/5)

Embedded SQL: intégrer le DDL, le DML et le Transaction Control à un langage de programmation :

- Déclarations d'objets ou d'instructions.
- Exécutions d'instructions.
- Gestions des variables et des cursors.
- Traitement des erreurs.

Instructions : DECLARE, TYPE, DESCRIBE, VAR, CONNECT, PREPARE, EXECUTE, OPEN, FETCH, CLOSE, WHENEVER.

# BLOC PL/SQL

Un bloc est composé de 3 sections :

DECLARE

(déclaration des variables, des constantes, des exceptions et des curseurs)

BEGIN

(instruction SQL, PL/SQL, structures de contrôle)

EXCEPTION

(traitement des erreurs)

END ;

Il est possible d'ajouter des commentaires à un bloc : --

Permet de mettre en commentaire ce qui suit sur la ligne.

/\*... ... \*/ : mettre en commentaire plusieurs lignes.

Des étiquettes (label) permettent de marquer des parties de bloc, sous la forme <<nometiquette>>.

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Agrégats (GROUP BY) (1/2)

Permettre des calculs statistiques sur des regroupements introduits par GROUP BY.

Commande : SELECT liste de colonnes, fonction groupe FROM S GROUP BY liste colonnes ;

Les colonnes projetées (derrière le SELECT) doit être identique à la liste de colonnes de regroupement (derrière le GROUP BY).

Les noms des colonnes peuvent apparaître dans des calculs élémentaires.

# Agrégats (GROUP BY) (2/2)

Exemple de fonctions de groupe :

AVG (coln) : Moyenne des valeurs de colonne.

COUNT (colonne) : Pour chaque regroupement, nombre de lignes regroupées où colonne est non NULL.

COUNT (\*) : Pour chaque regroupement, nombre de lignes regroupées.

MAX (colonne) : Valeur maximum de la colonne pour chaque regroupement.

MIN (colonne) : Valeur minimum de la colonne pour chaque regroupement.

SUM (coln) : Somme des valeurs de colonne pour chaque regroupement.

# VARIABLES

## Types scalaires

BINARY\_INTEGER  
DEC  
DECIMAL  
DOUBLE PRECISION  
FLOAT  
INT  
INTEGER  
NATURAL  
NATURALN  
NUMBER  
NUMERIC  
PLS\_INTEGER  
POSITIVE  
POSITIVEN  
REAL  
SIGNTYPE  
SMALLINT

CHAR  
CHARACTER  
LONG  
LONG RAW  
NCHAR  
NVARCHAR2  
RAW  
ROWID  
STRING  
UROWID  
VARCHAR  
VARCHAR2

## Types Composés

RECORD  
TABLE  
VARRAY

## Types Références

REF CURSOR  
REF type\_objet

## Grands objets

BFILE  
BLOB  
CLOB  
NCLOB

BOOLEAN

DATE

INTERVAL DAY TO SECOND  
INTERVAL YEAR TO MONTH  
TIMESTAMP  
TIMESTAMP WITH LOCAL TIME ZONE  
TIMESTAMP WITH TIME ZONE

# VARIABLES (Sous-types)

PL/SQL propose des sous-types synonymes des types donnés ci-dessus. Ils permettent d'assurer la compatibilité avec les types standards ANSI/ISO et IBM.

Type ORACLE	Sous-types
NUMBER	DEC, DECIMAL, NUMERIC, DOUBLE PRECISION, FLOAT, REAL INTEGER, INT, SMALLINT
BINARY_INTEGER	NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE
VARCHAR2	STRING, VARCHAR
CHAR	CHARACTER

Types définis par l'utilisateur

Syntaxe : SUBTYPE nom\_sous\_type IS type;

# OPERATEURS

- Arithmétique : + / \* ( )

Exemple : 1.15 \* PRIX  
(2 \* MTLIG)/5  
SYSDATE +15

- Sur les chaînes de caractères : concaténation ||

Exemple : 'Monsieur' || NOM

# CONDITIONS

- Comparaison simple :

expression1 {=,!=,<>, <,<=, >, >=} expression2

- Appartenance à un ensemble de valeurs

expression1 IN (expression2,...)

- Appartenance à un intervalle de valeurs

expression1 BETWEEN expression2 AND expression3

- Comparaison par rapport à un format de chaîne de caractères

expression1 LIKE 'format'

Le format peut inclure les métacaractères :

"%" pour désigner une suite de 0 à n caractères

"\_" pour désigner un et un seul caractère.

# CONDITIONS (opérateurs logiques)

Une expression NULL n'est ni VRAI ni FAUX.

- Négation d'une expression logique

NOT expression

- Combinaison d'expressions logiques

expression1 { AND / OR } expression2

# FONCTIONS (1/2)

- Fonctions scalaires mathématiques  
ABS(), ROUND(), MOD(), ...
- Fonctions scalaires chaînes de caractères  
CHR(), LENGTH(), LOWER(), ...
- Fonctions scalaires dates  
ADD\_MONTHS (d,n), SYSDATE

# FONCTIONS (2/2)

- Fonctions scalaires de conversion

CONVERT (char,destination[,source]),  
TO\_CHAR (caractère) ...

- Fonctions de comparaisons

NULLIF (expr1, expr2)

- Fonctions scalaires diverses

DECODE (colonne, valeur 1, resul1 [,valeur2, resul2 ...],[défaut])  
si colonne a la valeur valeur 1, la forme sera resul1.

# INSERT

## Syntaxe

```
INSERT INTO table [(colonne ,...)] VALUES  
(expression ,...) ;
```

## Exemple

Création d'un client (les colonnes non citées sont à NULL).

```
SQL> insert into CLIENTS (NOCLI, NOMCLI)  
      values (37, 'Ets LAROCHE');
```

# **DELETE**

## Syntaxe

```
DELETE FROM table [WHERE condition] ;
```

## Exemples

```
Delete from ARTICLES where REFART = 'ZZZZ';  
delete from ARTICLES where DESIGNATION like 'Ca%';  
delete from LIGCDES;  
delete from ARTICLES where QTESTK is null;
```

TRUNCATE : permet de supprimer toutes les lignes d'une table et de reprendre les conditions de stockage adoptées lors de la création de la table.

Ex : TRUNCATE table produits;

# UPDATE

## Syntaxe

```
UPDATE table SET colonne = expression, ...
[WHERE condition];
```

## Exemples

```
update ARTICLES set QTESTK=15 where REFART='AB10';
```

```
update ARTICLES set PRIX=PRIX*1.1 where REFART like 'AB%';
```

```
update ARTICLES set DESIGNATION=UPPER(DESIGNATION),
REFART=rtrim (REFART);
```

# TRANSACTIONS

Une transaction : ensemble d'instructions du DML (INSERT, UPDATE, DELETE) exécutées entre deux commandes CONNECT, COMMIT et ROLLBACK.

Validation : COMMIT

Annulation : ROLLBACK

Déclaration point de contrôle : SAVEPOINT xx

Un point de contrôle permet de mémoriser un état des données au cours de la transaction. La pose d'un SAVEPOINT permet d'annuler les modifications faites après la pose de celui-ci.

La transaction en cours est toujours active après la pose du SAVEPOINT.

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# VUE

## Syntaxe

```
CREATE [OR REPLACE] [FORCE | NO FORCE] VIEW nom[(colonnes,  
.....)] AS SELECT ..... [WITH CHECK OPTION | WITH READ ONLY];
```

FORCE | NOFORCE

Permet de forcer/d'empêcher la création de la vue si un des éléments sousjacent (table ou vue) n'est pas défini.

WITH CHECK OPTION

Vérifie, lors de l'insertion ou de la modification de lignes dans la vue, que les lignes insérées ou modifiées sont visualisables dans cette vue.

# VUES (1/3)

## Exemple

*Vue retournant les clients du département 44. L'option WITH CHECK OPTION empêche toute insertion de client n'appartenant pas à ce département.*

```
SQL> create or replace view V_CLI44 as
  2 select NOCLI, NOMCLI, CODE_POSTAL, VILLE from CLIENTS
  3 where CODE_POSTAL between 44000 and 44999
  4 with check option;
```

Vue créée.

```
SQL> insert into V_CLI44 values (255, 'ALAMBERT S.A.', 22000, 'ST BRIEUC') ;
*
```

ERREUR à la ligne 1:

```
ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```

# VUES (2/3)

## COMPILE

*Permet de valider la requête qui lui est associée et de détecter au plus tôt d'éventuelles références incorrectes dans celle-ci.*

*Utile après modification de la structure d'une des tables sous-jacentes de la vue. Si la compilation de la vue provoque une erreur, tous les autres objets de la base dépendant de cette vue deviennent invalides.*



# VUES (3/3)

## COMPILE

```
SQL> create or replace view V_CLICMD (NOCLIENT, NOM, COMMANDE,  
2 CP) as select CLIENTS.NOCLI, NOMCLI, NOCDE,  
3 CODE_POSTAL from CLIENTS, COMMANDES  
4 where CLIENTS.NOCLI = COMMANDES.NOCLI ;
```

```
SQL> create or replace view V_CLICMD44 as  
2 select * from V_CLICMD  
3 where CP between 44000 and 44999 ;
```

```
SQL> drop table COMMANDES cascade constraints ;  
Table supprimée.
```

```
SQL> alter view V_CLICMD compile ;  
Attention: Vue modifiée avec erreurs de compilation.
```

```
SQL> select * from V_CLICMD44 order by NOCLIENT ;  
select * from V_CLICMD44 order by NOCLIENT  
*
```

ERREUR à la ligne 1:

ORA-04063: view "GILLES.V\_CLICMD44" a des erreurs

# SYNONYM

Un synonyme est le nom alternatif donné à un objet TABLE, VIEW, SEQUENCE, SNAPSHOT, PROCEDURE, FUNCTION ou PACKAGE.

## Syntaxe

CREATE [PUBLIC] SYNONYM nom FOR objet ;

PUBLIC place le synonyme dans le schéma PUBLIC, le rendant ainsi visible de tout utilisateur défini sur la base de données (sans référence à un schéma). Sinon le synonyme reste local au SCHEMA de l'utilisateur propriétaire.

Suppression : DROP [PUBLIC] SYNONYM nom ;

# SEQUENCE (1/2)

La création d'un objet SEQUENCE met à disposition de l'utilisateur un générateur de nombres.

Utile en particulier pour la création de valeurs de clé primaire.

## Syntaxe

- CREATE SEQUENCE nom [paramètres];
- ALTER SEQUENCE nom paramètres ;
- DROP SEQUENCE nom ;

# SEQUENCE (2/2)

## PARAMETRES

START WITH n

Valeur initiale.

INCREMENT BY n

Pas d'incrémentation. Il peut être positif ou négatif.

MINVALUE n/NOMINVALUE

Valeur limite minimum ou non.

MAXVALUE n/NOMAXVALUE

Valeur limite maximum ou non.

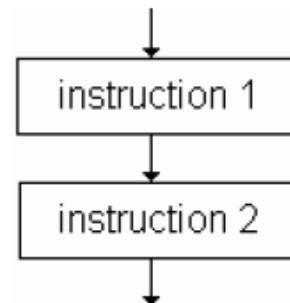
## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



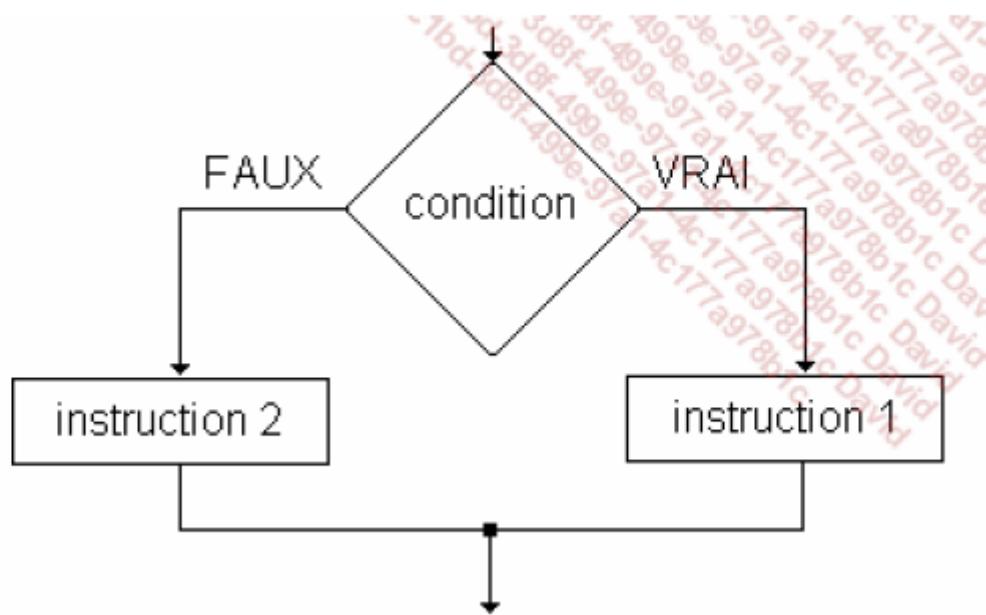
# Structures de contrôle (1/9)

- la séquence : exécution d'instructions les unes après les autres.



# Structures de contrôle (2/9)

- L'alternative : exécution d'instructions en fonction d'une condition.



# Structures de contrôle (3/9)

Les opérateurs utilisés dans les conditions sont les mêmes que dans SQL :

=; <, >, !, >=, <=, IS NULL, IS NOT NULL, BETWEEN, LIKE, AND, OR, etc.

## Exemple

*Si le client 10 est en cours de traitement, on le met à jour sinon on annule la transaction :*

```
if Vncli = 10 THEN
    UPDATE CLIENTS SET NOM = 'Dupont' where NOCLI = Vncli ;
    COMMIT ;
else
    ROLLBACK ;
end if ;
```

# Structures de contrôle (4/9)

*Exemple : comparer la valeur contenue dans la variable département avec chacune des valeurs qui suit le WHEN.*

```
declare
    département number:=44;
    libelle varchar2(40);
begin
    case département
        when 44 then libelle:='Loire Atlantique';
        when 49 then libelle:='Maine et Loire';
        when 53 then libelle:='Mayenne';
        when 72 then libelle:='Sarthe';
        when 85 then libelle:='Vendée';
        else libelle:='Hors Region';
    end case;
end;
```

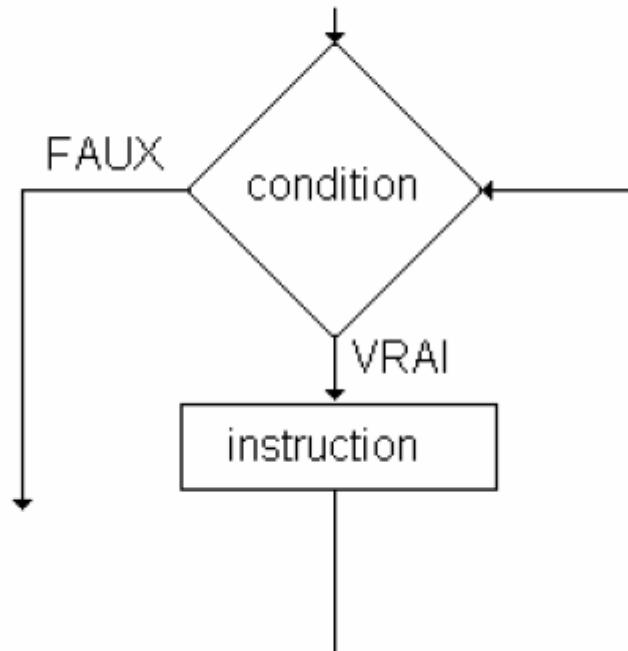
# Structures de contrôle (5/9)

*Exemple : imbriquer les instructions CASE les unes dans les autres.*

```
declare
    departement number:=44;
    region varchar2(80);
    libelle varchar2(80);
begin
    case
        when departement in (18,28,36,37,41,45) then
            region:='Centre';
        when departement in (44,49,72) then
            region:='Pays de la loire';
            case departement
                when 44 then libelle:='Loire Atlantique';
                when 49 then libelle:='Maine et Loire';
                when 72 then libelle:='Sarthe';
            else libelle:='Hors Region';
            end case;
        end case;
        dbms_output.put_line('region:'||region);
    end;
```

# Structures de contrôle (6/9)

- la répétitive : exécution d'instructions plusieurs fois en fonction d'une condition.



# Structures de contrôle (7/9)

```
[<<LABEL>>]
```

```
LOOP
```

```
instructions;
```

```
.....
```

```
END LOOP [LABEL] ;
```

Sortie de la boucle par la commande : EXIT [LABEL] [WHEN condition]

*Exemple : Insertion de 10 lignes dans CLIENTS :*

```
x := 0 ;
```

```
<<INCREMENTATION>>
```

```
LOOP
```

```
    x := x + 1 ;
```

```
    exit INCREMENTATION when x > 10 ;
```

```
    insert into CLIENTS (NOCLI) VALUES (x) ;
```

```
END LOOP INCREMENTATION ;
```

```
COMMIT ;
```

# Structures de contrôle (8/9)

[<<LABEL>>]

```
FOR indice IN [REVERSE] exp1..exp2 LOOP  
    instructions;
```

....

```
END LOOP [LABEL];
```

Sans l'option REVERSE, indice varie de exp1 à exp2 avec un incrément de 1.

Avec l'option REVERSE, indice varie de exp2 à exp1 avec un incrément de -1.

*Exemple : Crédation de 10 clients avec numéros successifs :*

```
FOR n IN 100..110 LOOP  
    insert into CLIENTS (NOCLI) values (n) ;  
END LOOP ;  
COMMIT ;
```

# Structures de contrôle (9/9)

```
[<<LABEL>>]  
WHILE condition LOOP  
    instructions;  
    .....  
END LOOP [LABEL];
```

La condition est une combinaison d'expressions au moyen d'opérateurs : <, >, =,  
!=, AND, OR, LIKE,..

*Exemple : Cration de 11 clients avec numeros successifs :*

```
x := 200 ;  
while x <= 210 LOOP  
    insert into CLIENTS (NOCLI) values (x) ;  
    x := x + 1 ;  
end loop ;  
COMMIT ;
```

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Variables (1/5)

2 sortes de variables :

- Les variables utilisateur, utilisées par SQL\*Plus ou pour de la substitution de texte dans des commandes SQL et SQL\*Plus avant leur exécution.
- Les variables de lien utilisées en interface avec le PL/SQL.

Syntaxe : DEF[INE] [variable = texte]

- Sans paramètre, DEFINE permet de visualiser les variables existantes.
- Les variables créées sont obligatoirement de type caractère.
- On peut définir 1024 variables au maximum.

# Variables (2/5)

## VARIABLE

Création d'une variable pour utilisation dans un bloc PL/SQL.

Syntaxe : VAR[TABLE] [nom [NUMBER/CHAR(n)]]

Sans paramètre, VARIABLE affiche les caractéristiques des variables existantes.

Une variable PL/SQL ne peut être valorisée que dans un bloc PL/SQL par des instructions spécifiques au PL/SQL, elle est considérée par le PL/SQL comme une variable de l'hôte (prefixée de :).

## PRINT

Affichage du contenu de la variable PL/SQL.

Syntaxe : PRINT variable

# Variables (3/5)

Variables ont des noms et des utilisations réservées :

\_EDITOR : Éditeur appelé par EDIT.

\_O\_VERSION : Version ORACLE.

\_O\_RELEASE : Numéro de "release" ORACLE.

\_CONNECT\_IDENTIFIER : Identifiant de la connexion.

\_DATE : Date courante.

UNDEFINE : Suppression d'une variable utilisateur.

Syntaxe : UNDEF[INE] variable

# Variables (4/5)

ACCEPT : Saisie d'une variable utilisateur.  
(Si la variable n'a pas été définie, elle est créée.)

Syntaxe :

ACCEPT variable [NUM/CHAR] [PROMPT texte] [HIDE]

## Exemple

*Saisie des variables V\_code et V\_nom :*

```
SQL> accept V_NOM
(saisie par l'utilisateur de : Dupont)
SQL> accept V_CODE NUM prompt "Code ?"
Code ? (saisie de : 15)
SQL> def
DEFINE V_NOM      = "Dupont" (CHAR)
DEFINE V_CODE     = 15 (NUMBER)
SQL>
```

# Variables (5/5)

Utilisation d'une variable de substitution

Syntaxe : &variable

Si la variable n'existe pas, une saisie est demandée.

```
SQL> accept NUMERO NUMBER prompt "Numero client ?"
Numero client ? 15
SQL> accept NOM prompt "Nom client ?"
Nom client ? TOTO
SQL> def tb = clients
SQL> select * from &tb where
  2> Nocli > &NUMERO and
  3> NOM = '&NOM';
ancien 1 : select * from &tb where
nouveau 1 : select * from client where
ancien 2 : nocli > &NUMERO
nouveau 2 : nocli >15
ancien 3 : NOM = '&NOM' ;
nouveau 3 : NOM = 'TOTO' ;
aucune ligne sélectionnée
SQL>
```

# Paramètres d'environnement

DEFINE {'&'/c/ON/OFF} : caractère utilisé pour les substitutions de variables.

EMBEDDED {OFF/ON} : chaque rapport débute sur une nouvelle page.

HEADSEP { |/c/ON/OFF} : caractère définissant un entête de colonne multiligne.

LINESIZE {80/n} : nombre de caractères par ligne.

NEWPAGE {1/n/NONE} : nombre de lignes entre le début de page et l'entête défini.

NUMFORMAT {format} : format par défaut des numériques.

NUMWIDTH {10/n}

# Paramètres d'environnement

Exemple :

```
set echo off
set feed off
set pagesize 60
set newpage 0
set linesize 80

rem Saisie du numero de commande

accept VCDE number prompt 'No de commande à éditer ? '

rem Mise en page

set term off

column REFART heading "Reference|article"
column DESIGNATION heading "Désignation"
column PRIXUNITHT heading "Prix H.T."

ttitle center "Confirmation de commande" -
skip 2 left "Numero : " VNOCDE -
skip 1 left "Client : " VNOM skip 3

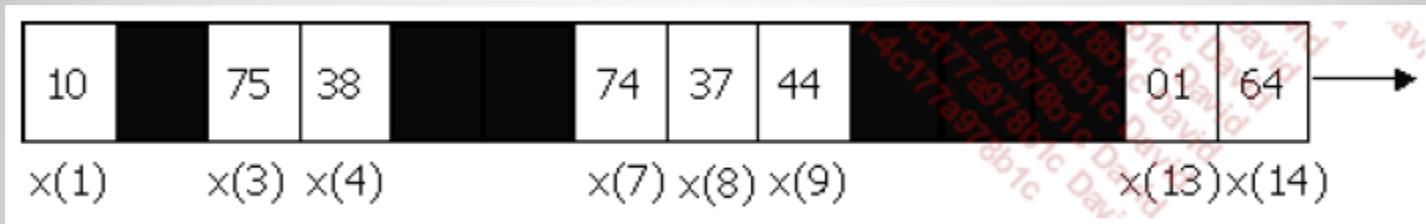
btitle tab 40 "Page " sql.pno

spool confcde.lis
```

# Structure de donnée (1/4)

Les collections de type nested table et Index by table

Ces collections sont de taille dynamique et il n'existe pas forcément de valeurs pour toutes les positions.



## Syntaxes

*Déclaration d'une collection de type nested table.*

TYPE nom\_type IS TABLE OF type\_element [NOT NULL]

*Déclaration d'une collection de type indexby table*

TYPE nom\_type IS TABLE OF type\_element [NOT NULL]  
INDEX BY BINARY\_INTEGER | VARCHAR2(n) | CHAR[(n)]

# Structure de donnée (2/4)

Les collections de type VARRAY

dimension maximale qui doit être précisée lors de la déclaration de la collection.

Ces collections possèdent une longueur fixe et donc la suppression d'éléments ne permet pas de gagner de la place en mémoire. Les éléments sont numérotés à partir de la valeur 1.

## Syntaxe

```
TYPE nom_type IS VARRAY (taille_maxi) OF type_element  
[NOT NULL]
```

# Structure de donnée (3/4)

## Les enregistrements

Ensemble de valeurs liées entre elles et regroupées sous un même nom.

Chaque valeur est stockée dans un champ. Par exemple, l'enregistrement qui permet de manipuler un client sera composé des champs numéro, nom, adresse...

Les enregistrements représentent donc un moyen simple de manipuler les informations qui sont liées entre elles.

Pour pouvoir créer un enregistrement, il est nécessaire au préalable, de définir le type d'enregistrement dans la section DECLARE du bloc PL/SQL.

# Structure de donnée (4/4)

## Syntaxe

```
TYPE nom_type IS RECORD ([nom_champs  
type_champs[[NOT  
NULL] := expression ]  
, ...]);  
nom_variable nom_type ;
```

type\_champs correspond à un type PL/SQL défini plus haut.

```
SQL> declare  
2      type T_CLIREC is record (  
3          NOCLI    number(4),  
4          NOM      char(20),  
5          ADR      char(20),  
6          CODPOST  number(5));  
7      UNCLIENT T_CLIREC ;  
8  begin  
9      -- Instructions  
10     UNCLIENT.NOCLI := 1024;  
11     -- Instructions  
12  end;  
13  /
```

# Variables définies dans un environnement extérieur à PL/SQL

Ces variables sont déclarées en dehors du bloc et utilisables dans le bloc.

Ce sont des champs d'écran créés dans Oracle\*Forms, ou des variables définies en langage hôte par les précompilateurs ou par SQL\*Plus (ces variables sont toujours prefixées de ":" dans un bloc PL/SQL).

```
SQL> variable x NUMBER
SQL> define t = CLIENTS
SQL> Begin
2> select COUNT(*) INTO :x from &t ;
3> end ;
4> /
Procédure PL/SQL terminée avec succès
SQL> print x
      x
      15
SQL>
```

## **PAUSE – REFLEXION**

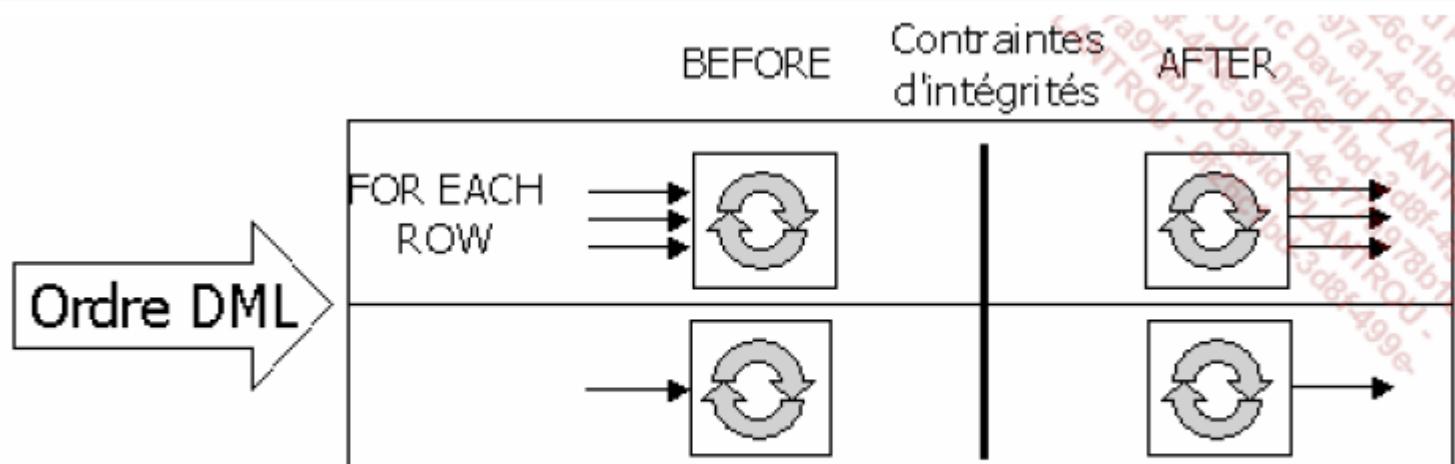
**Avez-vous des questions ?**



# Triggers Base de donnée (1/7)

**Trigger** : bloc PL/SQL associé à une table. Ce bloc s'exécutera lorsqu'une instruction du DML (INSERT, UPDATE, DELETE) sera demandée sur la table.

Remarque : Ne pas utiliser les triggers lorsqu'il est possible de mettre en place une contrainte d'intégrité (sont plus rapides).



*Exécution avant ou après vérification des contraintes d'intégrité pour chaque ligne ou chaque ordre.*

# Triggers - Présentation

- Utilisé pour implémenter des règles de gestion complexes et pour étendre les règles d'intégrité référentielles associées à la table lors de leur création de cette dernière.
- Son code est stocké en PL/SQL dans la base de donnée.
- Le déclenchement peut se propager en cascade, tout en respectant le principe d'atomicité d'une transaction.
- 2 sortes :
  - Triggers base de donnée
  - Triggers évènements

# Triggers Base de donnée (2/7)

Le bloc PL/SQL associé au trigger peut être exécuté pour chaque ligne affectée par l'ordre DML (option FOR EACH ROW), ou bien une seule fois pour chaque commande DML exécutée (option par défaut).

## → Triggers lignes / triggers de table

Remarques :

- Il est également possible de poser des triggers sur les vues (VIEW) afin d'intercepter les ordres DML que l'on peut y exécuter.
- Ces triggers permettent de maîtriser toutes les opérations qui sont effectuées sur les vues et pour l'utilisateur final, la vue est en tout point similaire à une table puisqu'il peut y faire les opérations INSERT, UPDATE et DELETE.

# Triggers Base de donnée (3/7)

## Syntaxe

```
CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE/AFFTER/INSTEAD OF}
{INSERT/UPDATE[OF col,...]/DELETE}
ON nom_table [FOR EACH ROW]
[FOLLOWS nom_autre_trigger[,...]]
```

### **[ENABLE/DISABLE]**

[WHEN (condition)]

Bloc PL/SQLv

### OR REPLACE

Remplace la description du TRIGGER s'il existe déjà.

### INSTEAD OF

*Le bloc PL/SQL qui suit remplace le traitement standard associé à l'instruction qui a déclenché le TRIGGER (pour une vue uniquement).*

# Triggers Base de donnée (4/7)

**Exemple :** Exécution d'un bloc PL/SQL après mise à jour de chaque ligne de la table ARTICLES si l'ancien prix est supérieur au nouveau :

```
create or replace trigger post_majprix
    after update of prix
    on articles
    for each row
    when (old.prix > new.prix)
declare
    ...
begin
    ...
end;
```

# Triggers Base de donnée (5/7)

## **Notion de trigger compose (Oracle 11) :**

À la différence d'un trigger simple dont l'instant de déclenchement est unique (avant ou après l'instruction ou la mise à jour de ligne),

un trigger composé peut comporter jusqu'à quatre sections correspondant chacune à un instant de déclenchement :

- Avant instruction,
- Avant chaque ligne,
- Après chaque ligne,
- Après instruction.

Avantage du trigger composé : les différentes sections peuvent partager des déclarations communes (variables, types, curseurs, sous-programmes, etc.).

# Triggers Base de donnée (6/7)

## Notion de trigger compose (Oracle 11) :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
    FOR {INSERT/UPDATE[OF col,...]/DELETE}
    ON nom_table
    [FOLLOWS nom_autre_trigger[, ...]]
    [ENABLE/DISABLE]
    [WHEN (condition)]
    COMPOUND TRIGGER
    [déclarations_communes]
    [BEFORE STATEMENT IS
        [déclarations_locales]
        BEGIN
        instructions
    END BEFORE STATEMENT]
```

# Triggers Base de donnée (7/7)

## Notion de trigger compose (Oracle 11) :

```
[BEFORE EACH ROW IS
    [déclarations_locales]
    BEGIN
        instructions
    END BEFORE EACH ROW]
[AFTER EACH ROW IS
    [déclarations_locales]
    BEGIN
        instructions
    END AFTER EACH ROW]
[AFTER STATEMENT IS
    [déclarations_locales]
    BEGIN
        instructions
    END AFTER STATEMENT]
```

# Triggers évènements (1/6)

## Triggers sur des événements système ou utilisateur :

- Suivre les changements d'état du système, tels que l'arrêt et le démarrage.
- Ces triggers vont permettre d'améliorer la gestion de la base et de l'application.
- Au niveau de l'utilisateur : triggers pour ses opérations de connexion et de déconnexion (logon et logoff), pour surveiller et contrôler l'exécution des ordres du DDL (CREATE, ALTER et DROP) et du DML (INSERT, UPDATE et DELETE).

# Triggers évènements (2/6)

## Exemple triggers sur des événements système :

STARTUP : déclenché lors de l'ouverture de l'instance.

SHUTDOWN : déclenché juste avant que le serveur ne commence le processus d'arrêt de l'instance.

SERVERERROR : déclenché lorsqu'une erreur Oracle se produit. (sauf pour les erreurs ORA-1034, ORA-1403, ORA-1422, ORA-1423 et ORA-4030 qui sont trop importantes pour que le processus puisse continuer à s'exécuter).

## Syntaxe

```
Create trigger nom_trigger {AFTER|BEFORE} evenement_systeme  
ON{DATABASE|SCHEMA}  
Bloc PL/SQL
```

# Triggers évènements (3/6)

## Exemple triggers sur des événements système :

EXAMPLE : les informations concernant chaque erreur provoquée sur le schéma de l'utilisateur qui crée le trigger sont enregistrées dans la table les erreurs qui a été créée au préalable.

```
SQL> create or replace trigger svr_erreur
  2  after SERVERERROR on schema
  3 begin
  4   insert into les_erreurs (utilisateur, temps, num_erreur)
  5   values (ora_login_user,sysdate,ora_server_error(1));
  6 end;
  7 /
```

Déclencheur créé.

```
SQL>
```

# Triggers évènements (4/6)

## Exemple triggers sur des événements utilisateurs :

AFTER LOGON : Après avoir établi une connexion avec le serveur.

BEFORE LOGOFF : Avant de rompre la connexion avec le serveur.

BEFORE CREATE, AFTER CREATE : Lorsqu'un objet est créé.

BEFORE ALTER, AFTER ALTER : Lorsqu'un objet est modifié.

BEFORE DROP, AFTER DROP : Lorsqu'un objet est supprimé.

## Syntaxe

```
Create trigger nom_trigger {AFTER|BEFORE}  
evenement_utilisateur ON{DATABASE|SCHEMA}  
Bloc PL/SQL
```

# Triggers évènements (5/6)

## Exemple triggers sur des événements utilisateurs :

Mise en place d'un trigger de surveillance des tables : dans l'exemple suivant, le trigger mis en place permet de scruter toutes les créations de table. Tous les renseignements sont stockés dans la table *INFOS\_TABLE* qui a été créée auparavant.

```
SQL> create or replace trigger srv_table
  2  after ddl
  3  on database
  4 begin
  5  if (ora_dict_obj_type='TABLE') then
  6    if (ora_sysevent='CREATE') then
  7      insert into infos_table(utilisateur,nom_table,creation)
  8      values (ora_login_user, ora_dict_obj_name, sysdate);
  9    end if;
 10   end if;
 11 end;
 12 /
```

Déclencheur créé.

```
SQL>
```

# Triggers évènements (6/6)

## Désactivation puis réactivation de triggers :

Syntaxe

```
ALTER TRIGGER nom_trigger {ENABLE|DISABLE};  
ALTER TABLE nom_trigger { ENABLE|DISABLE } ALL TRIGGERS;
```

```
SQL> ALTER TRIGGER bf_ins_commandes DISABLE;  
Déclencheur modifié.  
  
SQL> ALTER TRIGGER bf_ins_commandes ENABLE;  
Déclencheur modifié.  
  
SQL> ALTER TABLE commandes DISABLE ALL TRIGGERS;  
Table modifiée.  
  
SQL> ALTER TABLE commandes ENABLE ALL TRIGGERS;  
Table modifiée.  
  
SQL>
```

# Triggers – Remarques (1/4)

## Remarques :

- Les instructions de contrôle de transaction (ROLLBACK, COMMIT) ne sont pas autorisées.
- Triggers en cascade :
  - Un trigger peut provoquer le déclenchement d'un autre trigger.
  - ORACLE autorise jusqu'à 32 triggers en cascade à un moment donné (éviter les appels récursifs).
- Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites.

# Triggers – Remarques (2/4)

- Lors d'une insertion ou d'une modification, si un trigger peut potentiellement changer la valeur de NEW.colonne, il doit être exécuté avant l'événement (**BEFORE**). Sinon, la ligne aura déjà été insérée ou modifiée, et la modification de NEW.colonne n'aura plus aucune influence sur celle-ci.
- ERREUR ORA-0491 : accès à une table mutante (même avec un simple SELECT)

Une table mutante : table en cours de modification / une instruction DML INSERT, UPDATE ou DELETE, ou qui pourrait être modifiée en raison d'une contrainte DELETE CASCADE. Exemple : 2 tables A et B, B possédant une clé étrangère assortie d'une clause DELETE CASCADE pointant sur A. Lorsqu'une suppression est faite sur A, A et B sont considérées comme mutantes.

# Triggers – Remarques (3/4)

- Un Trigger peut répondre à plusieurs événements. Dans ce cas, il est possible d'utiliser les prédictats intégrés INSERTING, UPDATING ou DELETING pour exécuter une séquence particulière du traitement en fonction du type d'événement.

```
CREATE OR REPLACE TRIGGER myFirstTrigger
  AFTER UPDATE OR INSERT ON sadeg.emp
  FOR EACH ROW
  BEGIN
    DBMS_OUTPUT.ENABLE(20000);
    IF INSERTING THEN
      DBMS_OUTPUT.PUT_LINE(' INSERT ');
    END IF;
    IF UPDATING('ENAME') THEN
      DBMS_OUTPUT.PUT_LINE(' mis a jour ' || :NEW.ENAME);
    END IF;
  END;
```

# Triggers – Remarques (4/4)

- Ordre d'exécution des triggers :

Statement level before triggers

Row level before triggers

Row level after triggers

Statement level after triggers

- Possibilité sur un même évènement d'avoir plusieurs triggers.
- Il est également possible de prévoir la sequence d'enchainement des triggers avec les mots clés USE FOLLOWS et USE PRECEDES (Oracle 11.1).

Exemple : C follows B, et B follows A, alors C **follows** A (de manière indirecte).

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Curseurs (1/9)

Définition :

- Zone de mémoire de taille fixe, utilisé par le moteur de la base Oracle pour analyser et interpréter tout ordre SQL.
- Les statuts d'exécution de l'ordre se trouvent dans le curseur.

Syntaxe :

CURSOR nom curseur IS ordre\_select;

OPEN nom curseur;

FETCH nom curseur INTO liste\_variables;

*FETCH : ramène une seule ligne à la fois ;  
pour traiter n lignes, il faut une boucle.*

CLOSE nom curseur;

# Curseurs (2/9)

Exemple avec boucle FOR :

```
variable NO_CLIENT number
variable NOM_CLIENT char(20)
declare
    cursor c_cli is select NOCLI, NOMCLI from CLIENTS;
begin
    for V_CLI in C_CLI LOOP
        NO_CLIENT := V_CLI.NOCLI;
        NOM_CLIENT := V_CLI.NOMCLI;
    end loop ;
end;
```

Remarque : déclare implicitement la variable de parcours, ouvre le curseur, réalise les FETCH successifs et ferme le curseur.

# Curseurs (3/9)

Attributs d'un curseur :

%FOUND      Syntaxe : nom curseur%FOUND  
Booléen : TRUE si le dernier FETCH a ramené une ligne

%NOTFOUND    Syntaxe : nom curseur%NOTFOUND  
Booléen : TRUE si le dernier FETCH ne ramène pas 1 ligne

%ISOPEN      Syntaxe : nom curseur%ISOPEN  
Booléen : TRUE si le curseur est ouvert

%ROWCOUNT    Syntaxe : nom curseur%ROWCOUNT  
Numérique : #de lignes traitées par le dernier INSERT,  
UPDATE ou DELETE

# Curseurs (4/9)

Attributs d'un curseur :

tous les attributs ne sont pas testables n'importe comment.

Le tableau indique la valeur à laquelle on peut s'attendre lorsque l'on teste l'un de ces attributs.

	% ISOPEN	% FOUND	%NOT FOUND	%ROWCOUNT
OPEN avant après	FAUX VRAI	erreur NULL	erreur NULL	erreur 0
premier FETCH avant après	VRAI VRAI	NULL VRAI	NULL FAUX	0 1
FETCH avant après	VRAI VRAI	VRAI VRAI	FAUX FAUX	1 suivant les données
dernier FETCH avant après	VRAI VRAI	VRAI FAUX	FAUX VRAI	suivant les données suivant les données
CLOSE avant après	VRAI FAUX	FAUX erreur	VRAI erreur	suivant les données Erreur

# Curseurs (5/9)

ROWNUM :

retourne un numéro indiquant l'ordre dans lequel la ligne a été sélectionnée depuis une table. La première ligne sélectionnée porte le numéro 1, la deuxième le numéro 2...

```
SQL> select nocli, nomcli
  2  from clients
  3  where ROWNUM<=3;
```

NOCLI	NOMCLI
1	Albert
2	Bernard
3	Coutard

```
-----  
1 Albert  
2 Bernard  
3 Coutard
```

```
SQL>
```

Remarque : Si la commande SELECT, d'extraction des lignes, contient une clause ORDER BY, alors ROWNUM contient le numéro de la ligne avant le tri des données.

# Curseurs (6/9)

## Modification des valeurs d'un curseur

- La clause CURRENT OF permet d'accéder directement en modification ou en suppression à la ligne que vient de ramener l'ordre FETCH.
- Il faut au préalable réserver les lignes lors de la déclaration du curseur par un verrou d'intention (... FOR UPDATE OF nom\_col...).
- La clause CURRENT OF permet de ne pas manipuler la clé primaire.

# Curseurs (7/9)

## Modification des valeurs d'un curseur

### Exemple

Declare

```
CURSOR C1 is Select REFART, PRIXHT from ARTICLES  
FOR UPDATE OF PRIXHT ;  
ART C1%ROWTYPE ;
```

BEGIN

```
Open C1 ;  
fetch C1 into ART ;  
while C1%FOUND loop
```

```
    update ARTICLES set PRIXHT = ART.PRIXHT * 2  
        where current of C1 ;
```

```
    fetch C1 into ART ;
```

```
end loop ;  
commit ;  
close C1 ;
```

END ;

# Curseurs (8/9)

## Passage de paramètres

### Syntaxe

Déclaration du curseur avec paramètres :

```
CURSOR nom curseur( nom_paramètre type_donnée, ...)  
IS requête_SELECT;
```

Ouverture du curseur paramétré et passage de la valeur des paramètres :

```
OPEN nom curseur(valeur_parametre1, ...);
```

# Curseurs (9/9)

## Passage de paramètres

### Exemple

*Utilisation d'un curseur paramétré : dans l'exemple suivant, le curseur accepte en paramètre le numéro du client et permet de connaître les numéros des commandes passées par ce client.*

```
SQL> declare
  2      cursor c1(numero number) is
  3          select nocde, datecde
  4          from commandes
  5          where nocli=numero;
  6      vc1 c1%rowtype;
  7  begin
  8      -- ouverture du curseur
  9      open c1(1);
 10      -- traitement des information
 11      fetch c1 into vc1;
 12      -- fermeture du curseur
 13      close c1;
 14  end;
 15 /
```

Procédure PL/SQL terminée avec succès.

SQL>

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Procédures stockées

## Présentation

- Elles sont bien séparées du code serveur et peuvent être optimisées en parallèle par des spécialistes en SQL.
- Elles offrent un contexte d'exécution plus sécurisé.
- Possibilité de transmettre des informations plus complexe qu'en passant par une requête en dur dans le code.
- Définition exacte de ce que l'on doit donner en entrée et ce que l'on aura en retour.
- Côté serveur, le code les appelant n'est pas nécessairement plus concis mais est plus lisible.

# Procédures stockées (1/2)

## Définition

- Bloc de code PL/SQL nommé stocké dans la base de données et qui peut être exécuté à partir des applications ou d'autres procédures stockées.
- Dans un bloc PL/SQL, il suffit de référencer la procédure par son nom pour l'exécuter. À partir de l'outil SQL\*Plus, on peut utiliser l'instruction EXECUTE.

## Syntaxe

```
CREATE [OR REPLACE] PROCEDURE nom procédure  
[(paramètre {IN/OUT/IN OUT} type, ...)]  
{IS/AS} bloc PL/SQL ;  
OR REPLACE (Remplace la description si la procédure existe)
```

Paramètre	Variable passée en paramètre, utilisable dans le bloc.
IN	Le paramètre est passé en entrée de procédure.
OUT	Le paramètre renvoyé à l'environnement appelant.
Type	Type de variable (SQL ou PL/SQL).

# Procédures stockées (2/2)

## Exemple

Procédure de suppression d'un article :

```
create or replace procedure supp_art (numero in char) is
begin
    delete from ligcdes where refart=numero;
    delete from articles where refart=numero;
end;
```

Utilisation : EXECUTE supp\_art ('AB01') ;

# Fonctions stockées (1/2)

## Syntaxe

```
CREATE [OR REPLACE] FUNCTION nom de fonction  
[(paramètre [IN] type, ...)]  
RETURN type {IS/AS} Bloc PL/SQL ;  
OR REPLACE (La description est remplacée si la fonction existe)
```

Paramètre	Paramètre passé en entrée
Type	Type du paramètre (SQL ou PL/SQL).
RETURN type	Type de la valeur renvoyée par la fonction.

# Fonctions stockées (2/2)

## Exemple

Fonction factorielle :

```
CREATE FUNCTION factorielle (n IN NUMBER)
    RETURN NUMBER
    IS BEGIN
        if n = 0 then
            return (1) ;
        else
            return ((n * factorielle (n-1))) ;
        end if ;
    END;
```

# Procédures stockées

## REMARQUES :

- A la différence d'une procédure stockée, on ne peut pas appeler un trigger explicitement.
- Réduction du trafic sur le réseau (soumission d'un bloc PL/SQL au moteur au lieu d'une commande Sql).
- Inconvénient pour les développeurs : fragmentation du code entre la BD et l'applicatif.

# Package (1/6)

## Définition

regroupe logiquement des éléments PL/SQL liés, tels que les types de données, les fonctions, les procédures et les curseurs.

2 parties :

- Un entête ou spécification : décrire le contenu du package, de connaître le nom et les paramètres d'appel des fonctions et des procédures.
- Un corps (body) : contenir le code.

# Package (2/6) - Avantages

- Modularité : simplification relecture.
- Informations cachées : possibilité de rendre certains éléments invisibles à l'utilisateur du package.
- Performances : présent en mémoire dès le premier appel à un élément qui le compose. L'accès aux différents éléments du package est donc beaucoup plus rapide que l'appel à des fonctions et à des procédures indépendantes.

# Package (3/6)

## SYNTAXE DE L'ENTETE

```
CREATE PACKAGE nom_package AS
    -- Définition de type
    -- Déclarations de variables publiques
    -- Prototypes des curseurs publiques
    --Prototypes des PROCEDUREs et FUNCTIONs publiques
END [nom_package];
```

# Package (4/6)

## EXAMPLE

```
create or replace package GESTION_CLIENTS as
    type T_CLIREC is record (
        NOCLI number(4),
        NOM char(20),
        ADR char(20),
        CODPOST number(5),
        VILLE char(30));

    cursor C_CLIENTS return T_CLIREC;
    function CRE_CLI (NOM char, ADR char, CODPOST number,
                      VILLE char)
        return number;
    procedure SUPP_CLI (NOCLIENT number);
end GESTION_CLIENTS;
```

# Package (5/6)

## Corps du package

- Contient l'implémentation des procédures et fonctions exposées dans la partie entête.
- Contient des définitions de types et des déclarations de variables dont les portées sont limitées au corps du PACKAGE.
- Pour s'assurer que dans le corps du package tous les éléments précisés dans la spécification sont bien définis, PL/SQL effectue une comparaison point par point.

# Package (5/6)

## Syntaxe

```
CREATE PACKAGE BODY nom_package AS
```

Définitions de type locaux au package

Déclarations de variables locales au package

Implémentation des curseurs publics

Corps des PROCEDUREs/FUNCTIONs locales au package

Corps des PROCEDUREs et FUNCTIONs publiques

```
END [nom_package];
```

# Package (6/6)

## Exemple :

```
create or replace package body GESTION_CLIENTS as
    NOMBRE_CLI integer; -- Définition d'une variable locale

    -- Implémentation du curseur
    cursor C_CLIENTS return T_CLIREC is
        select NOCLI, NOMCLI, ADRCLI, CODE_POSTAL, VILLE
        from CLIENTS order by NOCLI;

    -- Fonction de création d'un nouveau client
    function CRE_CLI (NOM char, ADR char, CODPOST number , VILLE char)
        return number
    is
        NOUV_NOCLI number;
    begin
        select S_NOCLI.nextval into NOUV_NOCLI from DUAL;
        insert into CLIENTS
        values(NOUV_NOCLI, NOM, ADR, CODPOST, VILLE);
        NOMBRE_CLI := NOMBRE_CLI + 1;
        return NOUV_NOCLI;
    end;
```

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Gestion des erreurs (1/8)

## Structure d'un bloc PL/SQL :

```
DECLARE  
    -- déclaration des variables et des exceptions  
  
BEGIN  
    -- logique applicative  
  
EXCEPTION  
    -- traitement des exceptions  
  
END;
```

48f493  
ef499e97a1-4c  
01c David PLAN  
2d43d8-499e-97a1-4c  
a978b1c David PLAN  
TROU - 01  
e177a3

La section EXCEPTION : permet d'affecter un traitement approprié aux erreurs survenues lors de l'exécution du bloc PL/SQL.

On distingue 2 types d'erreurs :

- les erreurs internes Oracle,
- les anomalies dues au programme.

# Gestion des erreurs (2/8)

## Schéma d'un mécanisme d'interception des erreurs :

```
DECLARE  
    dépassement exception;  
    ...  
BEGIN  
    ...  
  
    DECLARE  
        dépassement exception;  
        BEGIN  
            ...  
            EXCEPTION  
            ...  
        END;  
    ...  
    EXCEPTION  
    ...  
END;
```

C:\177\...-4699-976  
AU 0128cf1bd-3  
Jawid PLANTROU  
de-grat-4c177a978bb1c L

La levée des exceptions ne permet pas de continuer normalement le traitement des opérations.

Pour contourner : s'appuyer sur la définition de sous-blocs au sein desquels les exceptions sont gérées. Possible alors d'exécuter une suite d'instructions.

# Gestion des erreurs (3/8)

## Règles à respecter :

- Définir et donner un nom à chaque erreur (différent pour les erreurs utilisateur et les erreurs Oracle).
- Associer une entrée dans la section EXCEPTION pour chaque nom d'erreur défini dans la partie DECLARE.
- Définir le traitement à effectuer dans la partie EXCEPTION.

## Exemple erreurs prédéfinies :

	Erreur Oracle	Valeur SQLCODE
INVALID_CURSOR	ORA01001	1001
TOO_MANY_ROWS	ORA01422	1422
VALUE_ERROR	ORA06502	6502

Remarque : Pour traiter les autres erreurs Oracle, il est possible d'utiliser le mot clé OTHERS et de connaître l'exception à l'origine de ce traitement en faisant appel aux fonctions SQLCODE et SQLERRM.

# Gestion des erreurs (4/8)

## Exemple :

Gestion de l'exception Oracle prédéfinie TOO\_MANY\_ROWS : dans l'exemple, la partie exception du bloc PL/SQL permet de traiter les exceptions qui correspondent à une tentative de stocker un ensemble de valeurs dans une seule variable (TOO\_MANY\_ROWS).

```
SQL> declare
  2      vnom clients.nomcli%type;
  3  begin
  4      select nomcli into vnom from clients;
  5  exception
  6      when too_many_rows then
  7          rollback;
  8  end;
  9 /
```

Procédure PL/SQL terminée avec succès.

```
SQL>
```

# Gestion des erreurs (5/8)

## Anomalies programme utilisateur :

Syntaxe

DECLARE

...

nom\_erreur EXCEPTION;

...

BEGIN

...

IF(anomalie)

    THEN RAISE nom\_erreur;

...

EXCEPTION

    WHEN nom\_erreur THEN

END;

*=> Sortie du bloc après exécution du traitement.*

# Gestion des erreurs (6/8)

**Exemple :** Par rapport à l'exemple précédent ( curseurs), on prévoit d'arrêter et d'annuler les modifications si un prix dépasse le plafond fixé.

```
DECLARE
    CURSOR C1 ...
    ART ...
    Depassement EXCEPTION ;
    NVPRIX NUMBER ;

BEGIN
    ...
    while C1%FOUND LOOP
        NVPRIX := ART.PRIXHT *2 ;
        if NVPRIX > 10000 then raise depassement ;
        update ARTICLES set PRIXHT = NVPRIX ...
        fetch ...
    end loop ;
    ...
EXCEPTION
    When depassement then rollback ;
END ;
```

# Gestion des erreurs (7/8)

**Utilisation de raise\_application\_error :** permet de définir des erreurs utilisateur depuis des sous-programmes.

*Exemple 1*

```
SQL> begin
 2      for vcli in (select nocli, ville from clients) loop
 3          if (vcli.ville='Nantes') then
 4              raise_application_error(-20001,'TROUVE');
 5          end if;
 6      end loop;
 7  end;
 8 /
begin
*
ERREUR à la ligne 1 :
ORA-20001: TROUVE
ORA-06512: à ligne 4

SQL>
```

# Gestion des erreurs (8/8)

**Utilisation de raise\_application\_error :** permet de définir des erreurs utilisateur depuis des sous-programmes.

Exemple 2

```
SQL> declare
  2      vnocde commandes.nocde%type;
  3  begin
  4      select nocde into vnocde
  5          from commandes
  6          where nocli=1;
  7  exception
  8      when no_data_found then
  9          -- il est possible de supprimer le client
 10          delete from clients where nocli=1;
 11      when others then
 12          raise_application_error (-20002,'Suppression impossible');
 13  end;
 14 /
declare
*
ERREUR à la ligne 1 :
ORA-20002: Suppression impossible
ORA-06512: à ligne 12

SQL>
```

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# REQUETES SQL COMPLEXES

Alias : Nom alternatif donné à une colonne ou à une table dans une requête.

- Changer le nom de la colonne à l'affichage ou pour la table résultante.
- Donner un nom comportant des caractères spéciaux (espace par exemple).

Exemple : Affichage d'un nom de colonne comportant des espaces

```
SQL> select NOCLI, NOMCLI as "Nom du client" from CLIENTS ;  
NOCLI Nom du client  
-----  
1000 DUPOND et DUPONT  
1001 DURAND et DURANT  
2 ligne(s) sélectionnée(s) .
```

# REQUETES SQL COMPLEXES

ANY: Compare, suivant l'opérateur donné (=,<>,<,<=,>,>=), les valeurs des colonnes spécifiées avec chacune des valeurs de la liste.

VRAI si au moins une des comparaisons est vraie.  
liste de valeurs = liste de constantes littérales ou des valeurs retournées par une sous requête.

Exemple :

```
select refart, designation, prix, qtestk from articles  
      where prix=ANY(select prix from articles  
                      where refart='ZZ01');
```

REFART	DESIGNATION	PRIX	QTESTK
CD21	Platine laser	500	20
ZZ01	Lot de tapis	500	25

---

CD21	Platine laser	500	20
ZZ01	Lot de tapis	500	25

# REQUETES SQL COMPLEXES

ALL : Compare, suivant l'opérateur donné (=,<>,<,<=,>,>=), les valeurs des colonnes spécifiées avec **chacune** des valeurs de la liste.

L'expression est vraie si toutes les comparaisons sont vraies.

Liste de valeurs : une liste de constantes littérales ou des valeurs retournées par une sous requête.

# REQUETES SQL COMPLEXES

**EXISTS:** La condition est vraie si la sousrequête retourne au moins une ligne.

## Syntaxe

```
SELECT ..... WHERE [()colonne [, colonne, ..]] EXISTS  
(SELECT ..../expression,...);
```

Exemple : La liste des clients n'est affichée que si au moins une commande existe dans la table COMMANDES.

```
SQL> select NOCLI, NOMCLI from CLIENT where exists (select 'x' from COMMANDES) ;
```

# **SOUS REQUETES IMBRIQUÉES**

Dans une sous requête imbriquée il n'y a pas de lien explicite entre la requête interne et la requête externe.

La requête interne est exécutée une seule fois pour construire la liste de valeurs, avant l'exécution de la requête externe (quel que soit le nombre de lignes ramenées par celle-ci).

Exemple :

```
select NOCLI, NOMCLI, VILLE from CLIENTS  
      where VILLE in (select VILLE from CLIENTS  
      where NOMCLI like 'DUBOIS%') ;
```

# SOUS REQUETES CORRÉLÉES

Dans une sous requête corrélée, la condition spécifiée dans la clause WHERE de la requête interne fait référence à une ou plusieurs colonnes de la requête externe.

La requête interne est donc réexécutée pour chaque ligne retornnée par la requête externe.

Exemple :

```
SQL> select nocli, nomcli from clients cl
      where not exists (select nocli from commandes co
                          where cl.nocli=co.nocli);
```

# PIVOTER LES DONNÉES (1/5)

ORACLE 11 : Possibilité de faire pivoter les données d'un résultat pour obtenir un affichage sous la forme d'un tableau croisé.

Exemple :

```
desc VENTES
```

Nom	NULL ?	Type
CODEPAYS	-----	CHAR (2)
CODEPROD	-----	CHAR (4)
ANNEE	-----	NUMBER (4)
MONTANT	-----	NUMBER (8)

# PIVOTER LES DONNÉES (2/5)

interroger la table pour produire un rapport donnant les ventes cumulées par pays et par année :

```
SQL> select ANNEE, CODEPAYS, sum(MONTANT)
      from VENTES group by ANNEE, CODEPAYS
      order by ANNEE, CODEPAYS;
```

ANNEE	CO	SUM (MONTANT)
-----	--	-----
2006	FR	1254781
2006	IT	1004852
2007	FR	1435278
2007	IT	986354
2007	US	835789
2008	FR	1451245
2008	IT	1034572
2008	US	1236745

# PIVOTER LES DONNÉES (3/5)

Pour afficher le résultat sous la forme d'un tableau croisé, il faut modifier l'écriture de la requête et utiliser la clause PIVOT.

```
SQL> select * from ( select ANNEE, CODEPAYS, MONTANT from VENTES )
      pivot
        sum(MONTANT)
        for CODEPAYS in ('FR','US','IT')
      )
      order by ANNEE;
ANNEE    'FR'      'US'      'IT'
-----  -----  -----
2006      1254781    1004852
2007      1435278    835789    986354
2008      1451245    1236745    1034572
```

Possibilité de mettre une clause AS pour renommer les colonnes

# PIVOTER LES DONNÉES (4/5)

clause UNPIVOT : permet de faire l'opération inverse : transformer un tableau croisé en tableau.

```
SQL> desc VENTES
```

Nom	NULL ?	Type
ANNEE		NUMBER(4)
FRANCE		NUMBER
USA		NUMBER
ITALIE		NUMBER

```
SQL> select * from VENTES;
```

ANNEE	FRANCE	USA	ITALIE
2006	1254781		1004852
2007	1435278	835789	986354
2008	1451245	1236745	1034572

# PIVOTER LES DONNÉES (5/5)

Exemple :

```
select * from VENTES  
unpivot (MONTANT for CODEPAYS in (FRANCE,USA,ITALIE)  
order by ANNEE,CODEPAYS;
```

ANNEE	CODEPA	MONTANT
-----	-----	-----
2006	FRANCE	1254781
2006	ITALIE	1004852
2007	FRANCE	1435278
2007	ITALIE	986354
2007	USA	835789
2008	FRANCE	1451245
2008	ITALIE	1034572
2008	USA	1236745

# COMMENTAIRES

Objectif : faciliter les opérations de mise à jour de la base.

Commentaires : sur les tables et les vues ainsi que sur chacune des colonnes qui composent ces tables et ces vues.

Etape indispensable car elle permet de connaître très exactement la signification et le rôle de chaque élément de la base de données.

Tous ces commentaires sont stockés dans un dictionnaire de données et sont accessibles en interrogeant certaines vues du dictionnaire.

## Syntaxe

```
COMMENT ON TABLE nom_table_vue IS 'texte';
```

```
COMMENT ON COLUMN nom_table_vue.nom_colonne IS 'texte';
```

# COMMENTAIRES

Objectif : faciliter les opérations de mise à jour de la base.

Commentaires : sur les tables et les vues ainsi que sur chacune des colonnes qui composent ces tables et ces vues.

Etape indispensable car elle permet de connaître très exactement la signification et le rôle de chaque élément de la base de données.

Tous ces commentaires sont stockés dans un dictionnaire de données et sont accessibles en interrogeant certaines vues du dictionnaire.

## Syntaxe

```
COMMENT ON TABLE nom_table_vue IS 'texte';
```

```
COMMENT ON COLUMN nom_table_vue.nom_colonne IS 'texte';
```

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# EXPRESSIONS REGULIERES

Outil pour travailler avec les chaînes de caractères

Pour pouvoir travailler avec les expressions régulières, Oracle propose un opérateur REGEXP\_LIKE et quatre fonctions : REGEXP\_INSTR, REGEXP\_SUBSTR, REGEXP\_REPLACE et REGEXP\_COUNT.

Les ancre

^ : marque le début de la ligne

\$ : marque la fin de la ligne

# EXPRESSIONS REGULIERES (1/6)

## Les quantificateurs

\* : correspond à 0 ou plusieurs caractères.

? : correspond à 0 ou 1 caractère.

+ : correspond 1 ou plusieurs caractères.

{m} : correspond à m caractères exactement.

{m,} : correspond à au moins m caractères.

{m, n} : correspond à au moins m caractères mais moins que n caractères.

# EXPRESSIONS REGULIERES (2/6)

Afin de permettre la construction d'expressions encore plus complexes, Oracle supporte les classes de caractères POSIX (*Portable Operating system Interface*).

Les classes de caractères POSIX :

[:alpha:]	caractère alphabétique.
[:lower:]	caractère alphabétique en minuscule.
[:upper:]	caractère alphabétique en majuscule.
[:digit:]	nombre.
[:alnum:]	caractère alpha numérique.
[:space:]	espace.
[:punct:]	caractère de ponctuation.
[:cntrl:]	caractère de contrôle non imprimable.
[:print:]	caractère imprimable.

## EXPRESSIONS REGULIERES (3/6)

Utilisation des classes de caractères POSIX dans les expressions régulières : les positionner entre des crochets [].

Par exemple, `[[:upper:]]` permet de rechercher une chaîne de caractères écrite uniquement en majuscules alors que `[[:lower:]]{5}` correspond à des mots de 5 lettres en minuscules.

Certains métacaractères ne possèdent pas le même sens suivant leur emplacement. C'est le cas pour les 2 métacaractères ^ et -.

Le caractère ^ lorsqu'il est placé en premier caractère d'une expression marque le début de l'expression par contre lorsqu'il apparaît comme premier caractère d'une liste de valeurs alors il marque la négation.

# EXPRESSIONS REGULIERES (4/6)

`^A[[:lower::]]` : correspond aux chaînes Ane, Anneau, Arbre, Appel, ....

`^A[^nop][[:lower:]]` : permet simplement d'extraire Arbre, Ami, Avant, ....

Le caractère permet - de simplifier l'écriture des plages de valeur en citant simplement le premier et le dernier caractères séparés par .

`^A[np][[:lower:]]` : permet d'identifier les chaînes suivantes Ane, Anneau, Appel, ....

Par contre s'il est placé en premier caractère d'une liste, il signale que les caractères situés à sa suite ne peuvent pas participer à la chaîne final.

`^A[nop][[:lower:]]` : permet d'identifier Avant, Ami, Arbre....

# EXPRESSIONS REGULIERES (5/6)

## REGEXP\_LIKE

Cet opérateur permet l'utilisation des expressions régulières pour la recherche dans les clauses WHERE ou bien lors de la construction des contraintes d'intégrité.

### Syntaxe

REGEXP\_LIKE(colonne, expression\_regulière)

Exemple : recherche les clients dont le nom commence par B suivi de caractères en minuscules.

```
Select nocli, nomcli from clients where regexp_like(nomcli,  
'^B[[:lower]]');
```

# EXPRESSIONS REGULIERES (6/6)

## REGEXP INSTR

Permet de localiser l'emplacement de départ d'une sous chaîne à l'intérieur d'une chaîne. L'avantage : il n'est pas nécessaire de citer la sous chaîne. Il suffit de la décrire à l'aide d'une expression régulière pour la localiser.

Syntaxe : REGEXP\_INSTR(chaîne, expression régulière)

## Exemple :

on souhaite connaître la position de la souschaîne S.A. dans le nom des clients.

```
SQL> select nomcli, regexp_instr(nomcli,'S.A.$') as position  
  2  from clients;  
  
NOMCLI                      POSITION  
-----  
DUPONT S.A.                  8  
Etb LABICHE                  0  
DUBOIS Jean                  0  
Bernard S.A.                 9  
Ets LAROCHE                  0  
DUBOIS Jean                  0  
LAROCHE                      0  
  
7 rows selected.  
  
SQL>
```

# Création d'un rapport au format HTML

Utilisation des commandes SET MARKUP et SPOOL.

## Syntaxe

```
SET MARK[UP] HTML [ON|OFF] [HEAD 'texte'] [BODY  
'texte'][TABLE 'texte'] [ENTMAP {ON|OFF}] [SPOOL  
{ON|OFF}] [PRE[FORMAT]{ON|OFF}]
```

HTML : (OFF par défaut) Positionner à ON la valeur de cette option permet d'activer la génération des résultats au format HTML.

HEAD : Personnaliser l'entête de la page HTML

BODY : Définir les attributs de la balise BODY.

TABLE : Permet de spécifier les attributs du tableau.

# Exemple rapport au format HTML

```
set echo on
set markup html on spool on entmap on -
head "<TITLE>Les commandes</TITLE>" -
<STYLE type='text/css'> -
<!-- BODY{background:#FFFFC6} --> -
</style>"-
BODY "TEXT=#FF00FF" -
TABLE "BORDER='2'" -
preformat off
column nocde heading 'N°'
column nocli heading 'Client'
column datecde heading 'Date'
column etatcde heading 'Etat' format A4
spool test.html
select nocde, nocli, datecde, etatcde from commandes;
spool off
set markup html off spool off
```

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# SQL DEVELOPPER

Application graphique. Permet d'exécuter des requêtes ou des scripts SQL, de gérer les objets d'une base de données (tables, vues, etc.) et de développer et mettre au point des programmes PL/SQL.

Oracle SQL Developer est gratuit et peut être téléchargé directement sur le site OTN (Oracle Technology Network).

<FAIRE PRÉSENTATION INTERACTIVE>

# Transactions (1/4)

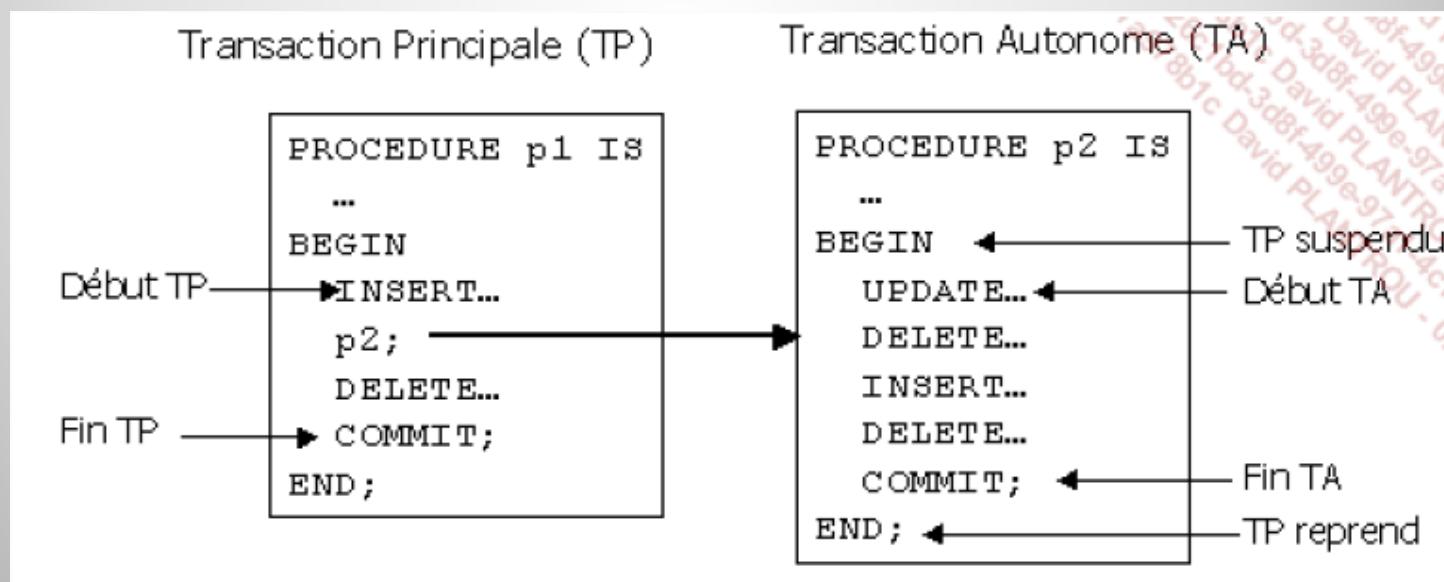
## Définition :

- Totalité des instructions de l'unité est exécutée avec succès soit aucune instruction n'est exécutée.
- La gestion des transactions est au cœur du traitement d'une base de données.
- Pour permettre à plusieurs utilisateurs d'y accéder simultanément, le SGBD doit gérer les **transactions avec le minimum de conflits** tout en veillant à la cohérence de la base de données.
- Une transaction est une unité logique composée d'une ou plusieurs instructions SQL.

# Transactions (2/4)

## Définition (suite) :

- Un transaction autonome est une transaction indépendante qui est lancée depuis une autre transaction : la transaction principale. Durant l'exécution de la transaction autonome, la transaction principale est suspendue.



# Transactions (3/4)

- Une transaction commence implicitement à l'exécution de la première commande SQL et se termine par un COMMIT pour validation ou un ROLLBACK pour annuler la transaction.
- La transaction poursuit l'exécution des commandes SQL une après l'autre jusqu'à ce que l'un des événements suivants se produit:
- COMMIT: toutes les modifications faites sur de la base de données jusqu'à ce point sont validées.
- ROLLBACK: toutes les modifications faites sur de la base de données jusqu'à ce point sont annulées.

# Transactions (4/4)

(suite)

- Instruction LDD (CREATE, DROP, RENAME, ou ALTER) : Oracle valide les instructions LDD courantes de la transaction et puis exécute et valide l'instruction LDD en question. Il s'agit d'un commit implicite.
- Fin normal du programme appellant: si le programme se termine sans erreurs, la transaction est validée implicitement (ex : EXIT sous sqlplus).
- Arrêt anormal du programme: tels qu'une coupure réseau ou un kill du programme, la transaction est annulée implicitement.

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# SQL Dynamique (1/5)

- Le SQL dynamique permet de construire dans un programme une requête SQL avant de l'exécuter. L'utilité principale est de fabriquer un code « générique » et réutilisable. Sans cela, le paramétrage d'une requête se limite aux valeurs de remplacement de la clause where :

```
CURSOR C_CURSEUR( LN$Id IN EMP.empid%TYPE ) IS
SELECT empno FROM EMP WHERE empid = LN$Id ;
FOR C_EMP IN C_CURSEUR( 1024 ) LOOP
.......
```

# SQL Dynamique (2/5)

## Exemple SQL dynamique :

```
CREATE PROCEDURE insert_into_table (
    table_name  VARCHAR2,
    deptnumber  NUMBER,
    deptname    VARCHAR2,
    location    VARCHAR2) IS
    stmt_str    VARCHAR2(200);

BEGIN
    stmt_str := 'INSERT INTO ' || table_name || ' values
    (:deptno, :dname, :loc)';

    EXECUTE IMMEDIATE stmt_str USING
        deptnumber, deptname, location;
END;
```

# SQL Dynamique (3/5)

## Avantages :

- Plus grande rapidité d'exécution : En moyenne, de 1.5 à 3 fois plus rapide.
- Support des types définis : tous les types définis par l'utilisateur, Objets, collections, etc...
- Support des types RECORD pour les ordres Select et une clause INTO

# SQL Dynamique (4/5)

## Exemple :

```
DECLARE
    Emp_id      emp.emp_id%TYPE := 1214 ;
    Emp_name    emp.name_id%TYPE ;
    Emp_rec     emp%ROWTYPE ;
    LC$Requete  VARCHAR2(256) ;
BEGIN
    LC$Requete:= 'SELECT emp_name from EMP WHERE empno = :1';
    EXECUTE IMMEDIATE LC$Requete INTO Emp_name USING Emp_id ;

    LC$Requete:= 'SELECT * from EMP WHERE empno = :1';
    EXECUTE IMMEDIATE LC$Requete INTO Emp_rec USING Emp_id ;
END ;
```

# SQL Dynamique (5/5)

## Autres Exemple :

```
requete:='insert into commandes values (:x, :x, :y, :z)';  
execute immediate requete using a,a,b,c;
```

```
EXECUTE IMMEDIATE 'UPDATE clients SET ville= :x' USING NULL ;
```

```
SQL> CREATE OR REPLACE PROCEDURE supprime  
 2    (type VARCHAR2,  
 3     nom VARCHAR2) AS  
 4 BEGIN  
 5   EXECUTE IMMEDIATE 'DROP ' || type || ';' || nom;  
 6 END;  
 7 /
```

Procédure créée.

```
SQL>
```

# Package DBMS\_SQL (1/1)

- En plus du SQL dynamique, Oracle (> 8.1.0) fournit le package DBMS\_SQL pour exécuter de façon dynamique des ordres SQL.
- Le package DBMS\_SQL est une librairie PL/SQL afin d'autoriser l'exécution d'ordres SQL construits dynamiquement.
- Le package DBMS\_SQL possède quelques avantages par rapport au SQL Dynamique :
  - il est supporté par les applications clientes,
  - il supporte la procédure DESCRIBE\_COLUMNS qui permet de connaître les informations relatives aux colonnes d'un curseur ouvert au travers de DBMS\_SQL,
  - il supporte la copie de données par blocs pour transférer les données issues d'une requête de type SELECT vers une collection.

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Collection (1/9)

- Toutes les références possèdent la même structure : le nom de la collection suivi d'un indice entre parenthèses.

## Exemple

*Déclaration et utilisation d'une collection :*

```
DECLARE
    TYPE liste IS TABLE OF VARCHAR2(15);
    lesnoms liste:=liste('B Martin','MP Macraigne','S Guegan',
    'T Groussard');
    i BINARY_INTEGER;
BEGIN
    ...
    IF lesnoms(i) ='G Viaud' THEN
        ...
    END IF;
    ...
END;
```

# Collection (2/9)

## Erreurs d'utilisation

```
DECLARE
    TYPE tableau IS TABLE OF INTEGER ;
    montab tableau ;
BEGIN
    montab(1) :=5 ;-- exception COLLECTION_Is_NULL
    montab :=tableau(10,5,3,6) ;
    montab(1) :=length('Hello') ;
    montab(2) :=montab(3)*2 ;
    montab('H') :=10 ;-- exception VALUE_ERROR
    montab(10) :=1 ;--exception SUBSCRIPT_BEYOND_COUNT ;
END ;
```

# Collection (3/9)

## Utilisation de tableau

Exemple :

```
SQL> CREATE TYPE LaCommande AS OBJECT(
 2      nocode number(9),
 3      montant number(10,2));
 4 /
Type créé.

SQL>
```

```
SQL> CREATE TYPE TabCommandes AS VARRAY(50) OF LaCommande;
 2 /
Type créé.

SQL>
```

# Collection (4/9)

## Utilisation de tableau

Exemple (suite) : Crédation de la table Facture

```
SQL> CREATE TABLE Facture (
  2      nofact number(9),
  3      nocli number(4),
  4      lesCommandes TabCommandes);
```

Table créée.

```
SQL>
```

# Collection (5/9)

## Utilisation de tableau

Exemple (suite) :

```
SQL> DECLARE
 2      nouvelles_cdes TabCommandes:=TabCommandes(
 3          LaCommande(3,600),
 4          LaCommande(10,800),
 5          LaCommande(11,1680));
 6 BEGIN
 7      ----- Ajout
 8      INSERT INTO Facture VALUES(
 9          1,1, TabCommandes(
10              LaCommande(2,500),
11              LaCommande(12,600),
12              LaCommande(13,1500)));
13      INSERT INTO Facture VALUES(
14          2,2, TabCommandes(
15              LaCommande(3,550),
16              LaCommande(10,135)));
17      -----> Modification
18      UPDATE Facture SET Lescommandes=nouvelles_cdes
19          where nofact=2;
20 END;
21 /
```

Procédure PL/SQL terminée avec succès.

SQL>

# Collection (6/9)

## Manipuler une ligne de collection (exemple INSERT)

```
SQL> DECLARE
  2      total number;
  3  BEGIN
  4      -----> Ajout d'infos
  5      INSERT INTO
  6          TABLE(Select stock from depot where nom='Paris')
  7          VALUES('VELO',2000);
  8      -----> Recherche d'infos
  9      SELECT SUM(montant) INTO total
 10         FROM TABLE( SELECT Lescommandes
 11                           FROM Facture
 12                         WHERE nofact=1);
 13  END;
 14 /
```

Procédure PL/SQL terminée avec succès.

```
SQL>
```

# Collection (7/9)

## Manipuler une ligne de collection (Bloc PL/SQL)

```
SQL> DECLARE
  2      stock_temoin TabStock:=Tabstock(
  3          TypeStock('AB21',200),
  4          TypeStock('Z201',300),
  5          TypeStock('UELO',550));
  6      ecart integer;
  7  BEGIN
  8      SELECT count(*) INTO ecart
  9      FROM TABLE(CAST(stock_temoin AS TabStock)) as temoin,
 10              TABLE(SELECT stock FROM depot WHERE nom='LYON') as test
 11      WHERE temoin.refart!=test.refart;
 12  END;
 13 /
```

Procédure PL/SQL terminée avec succès.

```
SQL>
```

# Collection (8/9)

## Méthodes

- EXISTS : booléen (test existence d'un élément)
- COUNT : nombre éléments d'une collection
- LIMIT : nombre maximum d'éléments possibles dans la collection.
- FIRST, LAST : le plus petit indice de la collection ainsi que le plus grand.
- PRIOR, NEXT : l'indice de l'élément précédent / suivant l'élément d'indice i dans la collection.
- EXTEND : modifier la taille d'une collection.
- TRIM : supprimer un ou plusieurs éléments à la fin de la collection.
- DELETE : supprimer un élément, plusieurs éléments ou la totalité d'une collection.

# Collection (9/9)

## Exemple d'utilisation de méthodes

```
i:=1  
WHILE(i<MesCommandes.COUNT) LOOP  
    IF MesCommandes.EXISTS(parcours) THEN  
        ...  
        i:=i+1;  
    END IF;  
    parcours:=parcours+1  
END LOOP;
```

```
i:=MesCommandes.FIRST;  
WHILE (i IS NOT NULL) LOOP  
    ...  
    i:=MesCommandes.NEXT(i);  
END LOOP;
```

Parcourir ts ls éléments d'une collection

## **PAUSE – REFLEXION**

**Avez-vous des questions ?**



# Autres fonction PL/SQL

- Copie des données par bloc
  - Commande FORALL
- Retour multi-lignes
  - Passage d'ensemble de lignes par paramètre des procédures / fonctions
- Wrap
  - Utilitaire pour coder le code source PL/SQL.
  - Possibilité de distribuer du code PL/SQL sans que les utilisateurs puissent avoir accès au code source.
  - Masquer l'algorithme mais pas les mots de passe ou le noms de table
- DBMS Output
  - Envoyer des messages depuis une procédure, une fonction, un package ou un déclencheur (trigger) de base de données.