

# Administration de bases de données Oracle

Cycle Ingénieur Applications Web et Mobiles -Semestre 08

**Ecole d’Ingénierie Digitale et d’Intelligence Artificielle**  
Université Euro-Méditerranéenne de Fès

Pr. Abderrahim El Qadi  
Département Mathématique Appliquée et Génie Informatique  
ENSAM, Université Mohammed V de Rabat

A.U. 2023/2024

## SQL Sous Oracle et PL /SQL

Partie 1 : SQL sous Oracle

1. Langage de Manipulation de Données (LMD) de SQL
2. Gestion de transactions
3. Les objets de schéma
  - Les Tables
  - Les Vues
  - Les Séquences
  - Les synonymes

Partie 2 : Langage PL/SQL

Langage PL/SQL

1. Procédures et fonctions stockées
2. Les Déclencheurs (Triggers)
3. Groupement de procédures et packages

## Partie 1 : SQL sous Oracle

1. Langage de Manipulation de Données (LMD) de SQL
2. Gestion de transactions
3. Les objets de schéma
  - Les Tables
  - Les Vues
  - Les Séquences
  - Les synonymes

## 1. Langage de Manipulation de Données (LMD) de SQL : sélection des tuples



**Langage SQL (Structured Query Language) : est un langage d'interrogation structuré de bases de données relationnelles**

- Les instructions SQL couvrent 4 domaines :
  - Langage de Manipulation de Données (LMD) : SELECT, UPDATE, INSERT, DELETE,
  - Langage de Définition de Données (LDD) : CREATE, DROP,
  - Langage de contrôle des transactions : COMMIT, ROLLBACK
  - Langage de contrôle de données : GRANT, REVOKE

- La requête de sélection du langage SQL permet d'interroger la base de données en composant **les projections, les restrictions, les jointures...**
- Le résultat d'une sélection est représenté sous la forme d'une table d'une ou plusieurs colonnes.

La syntaxe générale est :

```

SELECT      <clause d'unicité>  <Liste de colonnes>
FROM        <liste de tables>
WHERE       <critère de sélection :>
GROUP BY   <liste de colonnes>
HAVING      <critère de sélection>
ORDER BY   <critère d'ordre>
INTO [TEMP] <nom_de_table>

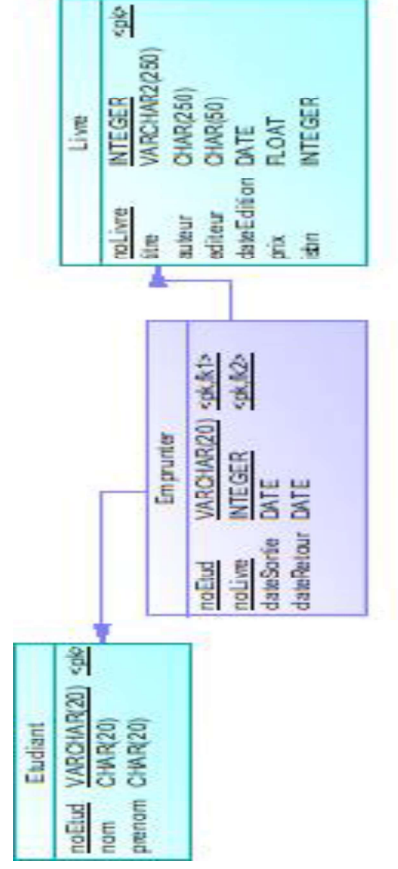
```

- Soit la base de données relationnelle ayant pour schéma :

Etudiant = (**noetud**, nom, prenom)

Livre = (**noliv**, titre, auteur, genre, prix)

Emprunt = (**noetud**, **noliv**, datesortie, dateretour)



- La **<clause d'unicité>** est définie au moyen des mots réservés "ALL" et "DISTINCT".
  - L'option "ALL" a pour effet de ne pas rejeter du résultat les tuples en double.
  - Lorsque "DISTINCT" est spécifié, un seul exemplaire des tuples en double est retenu dans le résultat de la requête.

Exemple :

```
SELECT DISTINCT (auteur) FROM Livre ;
```

- La **<liste de colonnes>** composée d'une étoile (c'est à dire "\*\*") indique que toutes les colonnes sont retenues. Dans le cas où les noms de colonnes sont donnés explicitement, il faut les séparer par des virgules.

Exemple :

```
SELECT * FROM Etudiant ;
```

Ou

```
SELECT noetud, nom, prenom FROM Etudiant ;
```

- La **<liste des tables>** est composée de noms de relations (ou vues) séparées par des virgules. Chaque nom peut être associé à un alias ;  
**Préfixer un nom de colonne est obligatoire lorsque plusieurs relations (ou vues) ont des noms de colonnes identiques. Cela permet d'éviter une ambiguïté de nommage.**
- Le **<critère de sélection>** "**WHERE**" est une expression booléenne vraie ou fausse. L'évaluation de ce critère conditionne le choix ou le rejet du tuple sur lequel il s'applique.

Exemple: SELECT \* FROM Livre l WHERE l.prix> 450

- Le **<critère d'ordre>** est constitué :

- d'une liste de noms de colonnes séparées par des virgules (c'est à dire *<liste de colonnes>* » ;
- d'une liste de numéros séparés par une virgule. Chaque numéro correspond au rang d'une colonne dans la liste de colonne de la clause" SELECT".

Exemple: SELECT \* FROM Livre l WHERE l.prix> 450 ORDER BY titre;

## 1.1 Les projections

<b>Projection :</b> $T = \Pi_{\langle A, B, C \rangle}(R)$	Consiste à définir un sous ensemble des colonnes de la liste de tables.
Où	Exemple : Projection sur "auteur" : <b>PROJ (livre/auteur)</b>
<b>T = PROJECT (R / A, B, C)</b> T ne contient que les attributs A, B et C de R.	<b>SELECT</b> auteur FROM livre ;

## 1.2 Les restrictions

<b>Restriction :</b> $T = \sigma_{\langle C \rangle}(R)$ Ou <b>T = RESTRICT (R / Condition)</b>	Correspond à un choix des tuples à sélectionner. Le résultat peut comporter zéro, un ou plusieurs tuples. Chacun des tuples sélectionné vérifie le critère de sélection.
T ne contient que les attributs de R qui satisfont la condition C	Exemple : <b>RESTRICT (livre/auteur = "CLAUDE")</b> <b>SELECT * FROM livre WHERE</b> auteur = 'CLAUDE' ;

Le critère de sélection prend une des formes suivantes :

*Exp* opérateur de comparaison **exp**  
**exp** (NOT] BETWEEN *exp1* AND *exp2*  
**exp** (NOT] IN (*liste de valeurs*)  
*Nom\_de\_colonne* [NOT] LIKE "*chaîne*"  
*Nom\_de\_colonne* IS [NOT] NULL "*chaîne*"

Où

- **exp** désigne une expression qui est un nom de colonne, une constante ou une combinaison des deux reliés par des opérateurs arithmétiques (+, -, \*, /).
- opérateur\_de\_comparaison peut être : =, <>, <, >, <=, >=.
- Les conditions peuvent être reliées par AND, OR, NOT et des parenthèses.

## Exemples :

a) SELECT titre, auteur, prix FROM livre WHERE prix - 200 >= 0 ;  
b) SELECT titre, auteur, prix \* (1-5/100) FROM livre WHERE prix >= 100 ;  
c) SELECT \* FROM livre WHERE prix >= 100 AND auteur = 'CLAUDE' ;  
d) SELECT \* FROM livre WHERE auteur = 'CLAUDE' OR auteur = 'DENALOY' ;  
e) SELECT \* FROM livre WHERE NOT (auteur = 'CLAUDE' OR auteur = 'DENALOY') ;  
f) SELECT \* FROM livre WHERE prix BETWEEN 80 AND 100;  
g) SELECT \* FROM livre WHERE prix IN (80, 90, 100);  
h) SELECT \* FROM Emprunt WHERE date retour IS NULL;

i) **SELECT \* FROM** livre **WHERE** titre **Like** "Algorithme %";

- **LIKE** '%a' : permet de rechercher toutes les chaînes de caractère qui se termine par un "a".
- **LIKE** 'a%' : ce modèle permet de rechercher toutes les lignes de "colonne" qui commence par un "a".
- **LIKE** '%a%' : ce modèle est utilisé pour rechercher tous les enregistrements qui utilisent le caractère "a".
- **LIKE** 'pa%on' : ce modèle permet de rechercher les chaînes qui commencent par "pa" et qui se terminent par "on", comme "pantaloon" ou "pardon".
- **LIKE** 'a\_c' : ce modèle permet de retourner les lignes "aac", "abc" ou même "azc".

Le caractère **"\_"** (underscore) peut être remplacé par n'importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage **"%"** peut être remplacé par un nombre incalculable de caractères).

– Les restrictions ordonnees

La clause **"ORDER BY"** permet de trier les tuples d'une requête selon un ou plusieurs critères

Exemples :

- a) **SELECT \* FROM** livre  
**ORDER BY** auteur, titre ;
- b) **SELECT** noliv, titre, auteur, genre, prix  
**FROM** livre  
**ORDER BY** 3,2 ;
- Les mots clés **ASC** (ordre croissant ; par défaut) et **DESC** (ordre décroissant) peuvent être placés après les critères d'ordre.
- c) **SELECT \* FROM** livre  
**ORDER BY** auteur **DESC**, titre;

## 1.3 Les fonctions

- **Fonction COUNT** : donne le nombre de tuples répondant à un critère.

Exemples:

```
a) SELECT COUNT (*) FROM livre;  
b) SELECT COUNT (DISTINCT auteur) FROM livre ;  
c) SELECT COUNT (DISTINCT auteur) FROM livre  
WHERE genre = 'INFORMATIQUE' ;
```

- **Fonction SUM** : donne la somme des valeurs d'une colonne numérique.

Exemples :

```
a) SELECT SUM (prix) FROM livre ;  
b) SELECT SUM (prix) FROM livre WHERE genre='INFORMATIQUE' ;
```

### – Groupement des données

La clause "**GROUP BY**" permet de partitionner une relation. Chaque partition possède une même valeur pour les attributs spécifiés après le terme "GROUP BY".

Elle est généralement utilisée conjointement avec les fonctions de calcul (COUNT, MIN, ...).

Exemples :

- a) Calculer le nombre de livres répertoriés pour chaque auteur.

```
SELECT auteur, COUNT (*) FROM livre  
GROUP BY auteur ;
```

- b) Donner pour chaque auteur, les prix des livres les plus chers et les moins chers.

```
SELECT auteur, MAX (prix), MIN (prix) FROM livre  
GROUP BY auteur ;
```



La clause "GROUP BY" peut être combinée avec "HAVING".

La clause "**HAVING**" sélectionne un sous ensemble de groupe en fonction d'un critère de la même manière que la clause "WHERE" sélectionne un nombre de tuples.

c) Afficher le nom de l'auteur, le prix du livre le plus cher et le moins cher pour cet auteur, ceci pour les auteurs ayant écrit plus d'un livre.

```
SELECT  auteur, MAX (prix), MIN (prix) FROM livre
GROUP BY auteur
HAVING  count (*) > 1 ;
```

## 1.4 Sauvegarde du résultat dans une table temporaire

Le résultat d'une requête peut être rangé dans une table temporaire grâce à la clause "**INTO TEMP**".

Exemple :

```
SELECT auteur, MAX (prix) maxi, MIN (prix) mini
FROM livre INTO TEMP T1 ;
```

Les attributs de T1 sont : auteur, maxi, mini.

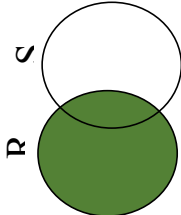
Où

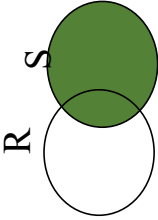
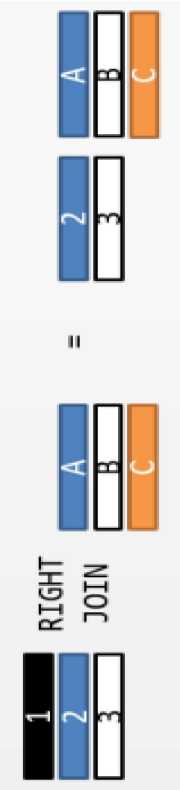
```
Create table T1 AS SELECT auteur, MAX (prix) maxi, MIN (prix) mini
FROM livre ;
```

## 1.5 Les jointures

Elles permettent de regrouper des informations provenant de plusieurs relations en fonction d'un critère de comparaison d'attributs.

<p><b>Produit cartésien</b></p> <p>Poduit cartésien : <b>T = R x S</b>  ou  <b>T = PRODUCT (R, S)</b></p> <p>Associe chaque tuple de R à chaque tuple de S : Select *  From R, S ;</p>	<p>Exemple : Sélectionner l'ensemble des livres empruntés :</p> <p><b>Join (livre, emprunt/livre.noliv=emprunt.noliv et dateretour = vide)</b></p> <p>SELECT l.noliv, titre, auteur FROM livre l, emprunt e  WHERE l.noliv=e.noliv AND dateretour IS NULL</p> <p>Ou</p> <p>SELECT noliv, titre, auteur FROM livre l  INNER JOIN emprunt e  ON l.noliv=e.noliv AND e.dateretour IS NULL;</p> <p>Ce type de jointure est appelé équi-jointure car la comparaison des deux attributs est effectuée avec l'opérateur d'égalité. Elle est la plus fréquemment utilisée. Cependant, l'utilisation des autres comparateurs comme "&gt;" "&lt;" "&lt;&gt;" "&lt;=" ou "&gt;=" est autorisée</p>
--	---

<p><b>LEFT JOIN : MINUS</b></p>  <p>Renvoie tous les enregistrements de la table de gauche (R) et les enregistrements correspondants de la table de droite (S). Le résultat est 0 enregistrement du côté droit, s'il n'y a pas de correspondance.</p>	<p><b>SELECT column_name(s)FROM R  LEFT JOIN S  ON R.column_name = S.column_name;</b></p> <div> <div>1</div> <div>2</div> <div>3</div> </div> <div> <div>LEFT JOIN</div> <div>=</div> <div>1</div> <div>2</div> <div>3</div> </div> <div> <div>A</div> <div>B</div> <div>C</div> </div> <div> <div>A</div> <div>B</div> </div> <p><b>Différence : T = R - S ou T = MINUS (R, S)</b></p> <p>R et S doivent avoir même schéma.</p> <p>Permet de retirer les données de la relation S existant dans la relation R.</p> <p><b>Exemple</b> : les livres qui ne sont pas empruntés</p> <p>Select noliv from Livre  <b>MINUS</b>  select distinct noliv from Emprunt;</p>
<p><b>RIGHT JOIN</b></p>	<p><b>SELECT column_name(s)FROM R  RIGHT JOIN S  ON R.column_name = S.column_name;</b></p>

 <p>Renvoie tous les enregistrements de la table de droite (S) et les enregistrements correspondants de la table de gauche (R).</p>	<div data-bbox="190 140 369 933">  </div> <p>Le résultat est 0 enregistrement du côté gauche, s'il n'y a pas de correspondance.</p>
<p><b>Union de requête</b></p> <p>Le langage SQL fournit la possibilité d'effectuer une union ensembliste entre les résultats de deux requêtes.</p> <p><b>Union : <math>T = R \cup S</math> ou <math>T = \text{UNION}(R, S)</math></b></p> <p>R et S doivent avoir même schéma.</p>	<p>&lt;Requête de sélection&gt; UNION &lt;requête de sélection&gt;</p> <p>Les données redondantes sont supprimées par défaut sauf si l'option "ALL" est indiquée.</p> <p>Elle peut comporter une seule clause "ORDER BY" située à la fin et se référant à la colonne par son numéro.</p> <p><b>Exemple:</b></p> <pre>SELECT auteur FROM livre UNION SELECT nom FROM Etudiant ORDER BY 1 ;</pre>

## 1.6 Les requêtes imbriquées

Les requêtes imbriquées sont utilisées dans une requête pour effectuer une comparaison avec des valeurs provenant d'une autre requête (sous requête). La sous requête est une requête de sélection entre parenthèse.

### Remarque :

Le résultat de la sous requête doit avoir une seule colonne sauf si la comparaison s'effectue avec le prédicat "EXISTS".

Une condition de sous requête peut être formulée selon l'une des possibilités suivantes :

```
WHERE exp opérateur_de_comparaison {ALL / [ANY / SOME]}
(<requête select>) WHERE exp [NOT] IN
(<requête select>)
WHERE [NOT] EXISTS (<requête select>)
```

**ALL** : la condition est vraie si la comparaison est vraie pour chacune des valeurs retournées.

**ANY** : la condition est vraie pour au moins une valeur retournée.

**IN** : la condition est vraie si la comparaison est vraie pour une des valeurs retournées.

**EXISTS** : envoi le booléen vrai ou faux. Si l'évaluation de la sous requête donne lieu à une ou plusieurs tuples, la valeur retournée est vraie ; sinon, elle est fausse.

## 1.7 LMD de SQL : mise à jour des données

### i. Insertion de tuples

La requête d'insertion ajoute un ou plusieurs tuples dans une relation.

Syntaxe :

```
INSERT INTO <nom d'une relation> (<liste de colonnes>) VALUES
(<liste de valeurs>)
Où
INSERT INTO <nom d'une relation> <liste de colonnes> <requête select>
```

<i>liste de colonnes</i>	Peut-être omise. Dans ce cas, chaque colonne du tuple de la relation concernée sera affectée.
<i>liste des valeurs</i>	doit contenir autant de valeurs que de colonne concernées par l'insertion
<i>requête select</i>	Peut remplacer la <liste de valeurs>. Le nombre et le type des colonnes choisis dans la <requête select> doivent correspondre avec celles de <liste de colonnes>

De plus, la <requête select> **ne doit contenir ni "ORDER BY" ni "INTO TEMP"**.

- L'insertion d'un tuple peut être rejetée :
- si les contraintes d'unicité sur les attributs ne sont pas respectées ;
- si une colonne déclarée "NOT NULL" (lors de la création de la table) n'est pas instanciée lors de l'insertion.

### Exemples :

a) INSERT INTO livre VALUES (2, 'Base de données, 'Claude', 'INFORMATIQUE', 360) ;

b) INSERT INTO Livre (nolivre, titre, prix) VALUES (100, 'Base données SQL', (SELECT AVG(prix) FROM livre)) ;

## ii. Modification de tuples

La requête de modification permet de changer les valeurs de la relation ou de la vue désignée dans <nom d'une relation ou d'une vue>.

Syntaxe :

```
UPDATE <nom d'une relation> SET <nom de colonne = exp,...>
[ ( <liste de colonne> ) * = ( <liste exp> ) ] [WHERE <critère de sélection> ]
```

La clause "WHERE" permet de réduire l'ensemble des tuples concernés par la modification.

La <liste exp> est composée soit d'expressions séparées par une virgule soit d'une requête SELECT ramenant une seule ligne et un nombre de valeurs correspondant à la <liste de colonnes>.

### Exemple :

a) Affecter au livre numéro 8 le double du prix du livre numéro 5.

```
UPDATE livre SET prix = (SELECT 2*prix FROM livre WHERE nolivre= 5)
WHERE nolivre=8 ;
```

### iii. Suppression de tuples

**DELETE FROM** <nom de relation> [WHERE <critère de sélection>]

En l'absence de la clause" WHERE" tous les tuples de la relation sont détruits.

ou

**TRUNCATE TABLE** table ;

Après suppression des lignes de la table, les blocs mémoire libérés sont réaffectés à la base, la table conserve son extension initiale.

#### Exemples :

a) Effacer tous les tuples de la relation "livre2020".

```
DELETE FROM livre2020 ;
```

b) Effacer tous les livres de poésie de la relation livre.

```
DELETE FROM livre WHERE genre = 'POESIE';
```

El Qadi

-27-

SQL & PL/SQL Oracle

### Atelier 1. Soit la base de données relationnelle ayant pour schéma :

Etudiant = (noetud, nom, prenom)

Livre = (noliv, titre, auteur, genre, prix)

Emprunt = (noetud, noliv, datesortie, dateretour)

1. Créer ce schéma dans la base de données Oracle, en utilisant l'outils **Oracle SQL Developer Data Modeler**

2. Interroger la base de données, en utilisant l'outils **SQL Developer**

R1. Lister les livres d'éditeur « Dunod » classés par ordre décroissant des prix

R2. Lister les livres d'éditeur « Dunod » et de prix > 200

R3. Lister les livres qui n'ont pas d'auteurs

R4. Lister le nombre de livre par éditeur

R5. Donner les livres apparus en 2020

R6. Donner pour chaque éditeur, les prix des livres les plus chers et les moins chers

R7. Lister noliv, titre, et date sortie des livres empruntés

R8. Lister le nombre d'emprunt de chaque livre.

R9. Trouver les titres des livres empruntés et le nom de leur emprunteur.

R10. Lister les livres les plus souvent empruntés, triés par ordre décroissant des nombres d'emprunts.

R11. Donner les noms et prénoms des étudiants qui n'ont pas emprunté aucun livre.

El Qadi

-28-

SQL & PL/SQL Oracle

## 2. Gestion de transactions

- Transaction : comprend une série de commandes SQL pour la mise à jour de la BD.
- Les SGBD permettent aux utilisateurs de gérer leurs transactions. Ils peuvent à tout moment :
  - Valider les transactions avec la commande EXIT ou QUIT (Fin de session de travail), validation automatique.
  - Valider la transaction en cours par la commande **COMMIT**. Les modifications deviennent définitives et visibles à tous les utilisateurs.
  - Annuler la transaction en cours par la commande **ROLLBACK**. Toutes les modifications depuis le début de la transaction sont alors défaites.

```
SQL> INSERT ...; }
SQL> DELETE ...; }
SQL> COMMIT;      toutes ces actions sont validées
SQL> UPDATE...;
SQL> INSERT INTO ...; }
SQL> ROLLBACK;    toutes ces actions sont annulées
```

### Atelier 2. Mise en œuvre des transactions

1. Créer la table Table1 (coll number) ;
2. Insérer dans chacune de deux fenêtres (SqlDeveloper et SQLPlus), une ou deux lignes différentes. Que voit-on des lignes insérées à partir de chacun des deux écrans ? Conclusion ?
3. Valider les modifications dans l'une des deux fenêtres puis lister le contenu de la table dans chacune des deux fenêtres. Conclure sur la portée de la validation.
4. Valider les modifications dans chacune des fenêtres. Insérer à nouveau quelques lignes dans chacune des fenêtres puis annuler les modifications dans une des deux fenêtres. Lister le contenu de la table dans les deux comptes et conclure sur la portée de l'annulation.
5. Vider le contenu de la table, et valider l'opération. Ajouter une clé à la table. Vérifier l'ajout de la clé dans les deux fenêtres.
6. Ajouter dans chaque fenêtre une ligne ayant la même valeur pour la clé. Que se passe-t-il ? Annuler l'insertion dans la fenêtre encore active. Que devient le blocage ?
7. Refaire les opérations du point précédent, mais en validant l'insertion au lieu de l'annuler. Expliquer ce qui s'est passé dans les deux cas (annulation et validation).



### 3. Les objets de schéma

- Les objets de schéma constituent les éléments de base qui composent une base de données Oracle et permettent aux utilisateurs de stocker, d'organiser et de manipuler les données de manière efficace.
- Chaque objet de schéma est associé à un propriétaire, qui est généralement un utilisateur de la base de données disposant des autorisations appropriées pour créer et gérer ces objets.

- Voici quelques informations importantes sur les objets de schéma en Oracle :

<b>Objets</b>	<b>rôles</b>
Tables	elles sont utilisées pour organiser et stocker les informations de manière structurée.
Vues	elles sont des requêtes SQL stockées qui permettent de visualiser les données d'une ou plusieurs tables
Index	ils index sont des structures de données utilisées pour accélérer la recherche et la récupération des données dans les tables.
Séquences	elles sont des objets de base de données utilisés pour générer des numéros de séquence uniques.
Procédures et fonctions stockées	elles sont des blocs de code PL/SQL stockés dans la base de données
Déclencheurs (triggers)	ils sont des codes PL/SQL qui sont automatiquement exécutés en réponse à certaines actions effectuées sur la base de données, comme l'insertion, la mise à jour ou la suppression de données dans une table.
Synonymes	ils sont des alias pour les objets de base de données, Ils sont utilisés pour simplifier les requêtes et les références d'objets.
Packages	ils sont des objets qui regroupent des procédures, des fonctions, des types de données et d'autres objets connexes en une seule unité logique.



### 3.1.Gestion des Tables

#### 1. Création

```
CREATE TABLE <nom de la table> (
  < Nom_colonne> <type de données>, ... );
Ou :
CREATE TABLE <nom de la table> (
  < Nom_colonne> <type de données>, ... )
AS SELECT < nom de champs>,
FROM <nom de table> WHERE < prédicat> ;
```

#### Exemple :

```
CREATE TABLE Employe (
  NoEmp      CHAR (4) primary key,
  Nom        VARCHAR2 (30),
  Sal        NUMBER (8, 2) check (Sal>0),
  DateEmb    DATE);
```

### 2. Types de données

Les types de données manipulés par SQL sont identiques aux types simples des langages de haut niveau :

CHAR (n) : chaîne de caractères (1 à 32767)  
VARCHAR2(n) : longueur variable, n représente le maximum.  
SMALLINT : entier sur 2 octets (-32768 à 32767)  
INTEGER : entier sur 4 octets ( $-2^{32}$  à  $2^{32}$ )  
NUMBER (m) : entier à m chiffres  
NUMBER (n1, n2) : Réel à n1 chiffres au total (virgule comprise), n2 après la virgule  
SMALLFLOAT : DECIMAL (8)  
FLOAT : DECIMAL (16)  
DATE : date.

### 3. Contraintes d'intégrité

La définition des contraintes d'intégrité permet d'assurer le maintien de la cohérence des données de la base.

#### a. Contraintes sur table :

Unicité : UNIQUE (liste de colonnes) ;
Clé primaire : PRIMARY KEY (liste de colonnes) ;
Clé étrangère : FOREIGN KEY (liste de colonnes) REFERENCES nom_table (liste de colonnes) (ON DELETE CASCADE;) ;

- FOREIGN KEY (liste de colonnes) spécifié les colonnes qui composent la clé étrangère ;
- REFERENCES nom\_table (liste de colonnes) [ON DELETE CASCADE], définit la clé primaire référencée en tant que clé étrangère dans la contrainte d'intégrité référentielle.

Avec la clause ON DELETE CASCADE, dans le cas d'une suppression d'une occurrence de la table primaire, les occurrences liées sont automatiquement supprimées de la table secondaire.

El Qadi	-35-	SQL & PL/SQL Oracle
---------	------	---------------------

#### b. Contraintes sur colonnes :

Valeur obligatoire ou facultative : NOT NULL   NULL ;
Unicité : UNIQUE Vérifié que toutes les valeurs sont différentes
CHECK : Vérifié la condition précise
DEFAULT : Précise une valeur par défaut
CONSTRAINT : Permet de nommer une contrainte, ce nom est automatiquement affiché par le SGBD dès que la contrainte n'est pas vérifiée

#### Exemple de description de table avec contraintes :

CREATE TABLE Employe (		
NoEmp	NUMBER (4) NOT NULL,	
Nom	VARCHAR2(30),	
DateEmb	Date,	
Salaire	NUMBER (8,2),	
NoDept	NUMBER (3),	
	CONSTRAINT cle_pri PRIMARY KEY (NoEmp),	
	CONSTRAINT cle_etr FOREIGN KEY (NoDept) REFERENCES DEPARTEMENT	
(NoDept),		
	CONSTRAINT date_ok CHECK (DateEmb <= SYSDATE) );	

El Qadi	-36-	SQL & PL/SQL Oracle
---------	------	---------------------

#### 4. Modification de la structure d'une table

- **ALTER** : Ne fonctionne que sur les tables. Elle permet de rajouter, modifier ou supprimer des colonnes à une table. Elle peut être utilisée avec les mots clés ADD, MODIFY et DROP selon l'action à effectuer.

- **ADD** : ajoute la colonne en fin de table

```
ALTER TABLE nom-de-la table ADD (nom-de-colonne type-de-  
données , ...);
```

Exemple : ALTER TABLE Departement ADD (AdrEmail CHAR (15)) ;

- **MODIFY** : permet de modifier les types de données

Exemple :

```
ALTER TABLE Departement MODIFY (AdrEmail Varchar(30)) ;
```

- **DROP** : supprime les colonnes des tables existantes

```
ALTER TABLE nom-de-la table  
DROP COLUMN nom-de-colonne ;
```

On ne peut pas modifier ou supprimer une colonne si :

- elle est présente dans une vue ;
- elle sert dans un index ;
- une contrainte y fait référence.

- **Ajout de contrainte :**

```
ALTER TABLE nom-de-la table ADD CONSTRAINT nom_contrainte  
définition_contrainte ;
```

Exemple : ALTER TABLE Employe

```
ADD CONSTRAINT sal_ok CHECK (salaire>0);
```

- **Suppression de contrainte :**

```
ALTER TABLE nom-de-la table  
DROP CONSTRAINT nom_contrainte ;
```

Exemple : ALTER TABLE Employe

```
DROP CONSTRAINT sal_ok;
```

- **Désactivation une contrainte :**

Pour désactiver une contrainte sans la supprimer, ni la recréer, on utilise l'instruction ALTER TABLE avec la clause DISABLE.

```
ALTER TABLE nom_table  
DISABLE CONSTRAINT constraint [CASCADE];
```

- . La clause CASCADE désactive les contraintes d'intégrité dépendantes.
- . Lorsqu'une contrainte de clé primaire ou unique est désactivée, l'index unique est supprimé.

- **Activation une contrainte :**

Pour activer une contrainte sans la supprimer, ni la recréer, on utilise l'instruction ALTER TABLE avec la clause ENABLE.

```
ALTER TABLE nom_table  
ENABLE CONSTRAINT constraint;
```

- . Si on active une contrainte de clé UNIQUE ou PRIMARY KEY, un index correspondant est automatiquement créé.

## 3.2. VUE (VIEW)

### 1. Définition

- Une vue est une table virtuelle, et définie comme une sélection de données sur les tables de base.
- Les vues peuvent être interrogées comme des tables, ce qui peut simplifier les requêtes complexes, fournir un niveau supplémentaire de sécurité et abstraire la structure sous-jacente de la base de données.
- Les vues peuvent être interrogées avec d'opérations de manipulation des données (comme INSERT, UPDATE, DELETE) directement dessus à moins que ces vues ne sont pas créées avec la clause WITH CHECK OPTION.

### 2. Création d'une vue

```
CREATE [OR REPLACE] VIEW nom_vue AS  
SELECT colonne1, colonne2, ...  
FROM nom_table  
WHERE condition ....  
[WITH CHECK OPTION].
```

- ‘OR REPLACE’ est un mot-clé facultatif qui permet de remplacer une vue existante portant le même nom.
- La clause CHECK OPTION définit des contraintes d'intégrité référentielle à respecter lors des modifications de la base de données.
- La clause WITH READ ONLY interdit toute modification de données en utilisant le nom de la vue dans un ordre INSERT, UPDATE ou DELETE.

Exemple:

```
CREATE VIEW vue_Livre AS  
SELECT * FROM Livre  
WHERE editeur = 'Dunod';
```

L'utilisateur peut ensuite écrire ses propres requêtes sur la vue.

```
SELECT NOLIVRE, TITRE, PRIX FROM vue_Livre WHERE prix>=300;
```

```
INSERT INTO vue_Livre (nolivre, titre, prix, editeur) VALUES (110, 'SQL  
Sous Oracle', 350, 'Dunod');
```

```
select nolivre,prix,editeur from vue_livre; } Insertion dans les deux tables  
select nolivre,prix,editeur from livre; }
```

```
INSERT INTO vue_Livre (nolivre, titre, prix) VALUES (120, 'SQL', 450);
```

```
select nolivre,prix,editeur from vue_livre; } Insertion seulement dans la  
select nolivre,prix,editeur from livre; } table Livre
```

```
delete from vue_livre where nolivre=110; Suppression dans les deux tables
```

### 3.3. Les SEQUENCES

- Ce sont des objets permettent de :
- générer des clés primaires entières uniques dans des tables.
- avoir un compteur à titre informatif, que l'on incrémente quand on veut.

Exemple :

noLivre	Titre
10	POO en Java
20	Langage SQL
30	Administration Oracle

# 1. Création d’une définition de séquence

```
CREATE SEQUENCE nom_séquence  
[(MINVALUE x4 | NOMINVALUE)]  
[START WITH x2]  
[INCREMENT BY x1]  
[(MAXVALUE x3 | NOMAXVALUE)]  
[Cycle | NOCYCLE];
```

- MINVALUE : le minimum, NOMINVALUE (valeur min croissante à 1) ;
- START WITH : Valeur de départ du numéro de séquence ;
- INCREMENT : est optionnel (valeur par défaut est 1) ;
- MAXVALUE : limite de la série, NOMAXVALUE (valeur max croissante à 10<sup>27</sup>) ;
- CYCLE : option permet à la série de continuer même lorsque le maximum a été atteint. Dans ce cas la série suivante qui sera générée est celle correspondant à la valeur minimale.
- NOCYCLE : valeur par défaut, interdit à la série de produire des valeurs au-delà des max ou min.

## 2. Utilisation

Une séquence peut être appelée dans un ordre Select, Insert ou Update en tant que pseudo-colonne par :

- **Nom\_séquence.NEXTVAL** , qui génère à chaque référence la valeur suivante ( ou la première valeur lors de la première référence) ;
- **Nom\_séquence.CURRVAL**, qui donne la valeur courante du numéro de séquence à condition qu’un numéro ait été généré par un appel à NEXTVAL au cours de la même session.

Exemple:

-creation CREATE sequence es_employe START WITH 1000 INCREMENT BY 10 NOMAXVALUE;	- Utilisation: <ul style="list-style-type: none"><li>• INSERT Into Employe (NoEmp) values (es_employe.NEXTVAL); #valeur : 1000</li><li>• INSERT Into Employe (NoEmp) values (es_employe.NEXTVAL); #valeur : 1010</li><li>• INSERT Into Employe (NoEmp, nom, adresse) Values (es_employe.CURRVAL, ‘DUPOND’, ‘NICE’); # valeur : 1010</li></ul>
--	---

### 3. Modification

Il est possible de redéfinir certains paramètres d'un générateur de numéros de séquence par :

```
ALTER SEQUENCE nom_séquence  
INCREMENT BY ...  
.....  
;
```

### 3.4.Synonyme

- Un synonyme est un alias ou un nom alternatif pour un objet de base de données, tel qu'une table, une vue, une séquence, une procédure stockée, etc.
- Les synonymes permettent de simplifier les requêtes en utilisant des noms plus courts ou plus significatifs, d'accéder à des objets situés dans d'autres schémas sans spécifier le nom complet du schéma, et de masquer la structure physique des objets.



## Création

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM nom_synonyme FOR  
objet;
```

- ‘PUBLIC’ est un mot-clé facultatif qui indique que le synonyme est public et peut être utilisé par tous les utilisateurs de la base de données. Si ce mot-clé n'est pas spécifié, le synonyme est privé et ne peut être utilisé que par le propriétaire.
- ‘objet’ est l'objet de base de données pour lequel le synonyme est créé (table, vue, séquence, procédure stockée, etc.).

Exemple:

```
CREATE SYNONYM emp FOR hr.employees;  
utilisant le nom plus court emp au lieu de hr.employees  
SELECT * FROM emp;
```

## 3.5. Suppression des objets

**DROP** : Suppression des objets

```
DROP TABLE <nom-de-table>;
```

```
DROP VIEW <nom-de-la-vue>;
```

```
DROP SYNONYM <nom-de-synonym>;
```

```
DROP SEQUENCE nom_séquence
```

Il est parfois possible dans certains SGBD de renommer une table. C'est le cas d'Oracle avec l'ordre :

```
RENAME <ancien-nom> TO <nouveau-nom>;
```

### Atelier 3.

Ecrire en langage SQL les requêtes suivantes :

R1. Créer la vue `vue_LivreDunod` basée sur la table `Livre`, contenant toutes les informations des livres de l'éditeur 'Dunod'. Les noms de colonnes sont :

`NOLIVRE`, `TITRE`, `PRIX`, `EDITEUR`

- Afficher la structure et le contenu de la vue `vue_LivreDunod`.
  - Dans la vue `vue_LivreDunod` tenter de modifier le prix d'un livre
  - Ajouter un enregistrement dans la vue `vue_LivreDunod`, d'éditeur 'Eyrolles'
  - Vérifier l'insertion dans la table `Livre`, et la vue `vue_LivreDunod`
  - Supprimer ce dernier enregistrement ; et vérifier la suppression dans la table `Livre`
- R2. Créer une séquence nommée « `Livre_SEQUENCE` » à utiliser avec la colonne 'NOLIVRE' de la table `Livre`. Définir pour cette séquence la valeur de départ 50, une valeur maximale de 70 et un pas d'incrément de 10.
- Insérer trois enregistrements dans la table `Livre`. Utiliser la séquence créée pour remplir la colonne « `Nolivre` ». Vérifier les ajouts.
  - Insérer le quatrième enregistrement dans la table `Livre`. Que s'est-il passé ?

### Partie 2 : Langage PL/SQL

1. Langage PL/SQL
2. Procédures et fonctions stockées
3. Les Déclencheurs (Triggers)
4. Groupement de procédures et packages

# 1. Langage PL/SQL

Le langage PL/SQL (Procedural language/SQL) est une extension du langage SQL qui offre un environnement procédural au langage SQL.

## 1.1 Caractéristiques de PL/SQL

- Dans l'env. PL/SQL, les ordres SQL et PL/SQL sont regroupés en blocs. Chaque bloc comporte trois sections :

```
DECLARE (Facultatif)
    Variables, curseurs, exceptions utilisateur
BEGIN (Obligatoire)
    Ordres SQL
    Instructions PL/SQL
EXCEPTION (Facultatif)
    Actions à exécuter en cas d'erreur
END; (Obligatoire)
```

## 1.2 Les variables

L'ensemble des instructions du langage utilisent des variables pour réaliser les traitements.

### 1.2.1 Déclaration

PL/SQL offre deux classes de types de données : scalaire et composé :

- **Types scalaires** : ce sont les types définis dans la section de création de tables.
- La déclaration d'une variable se fait sous la forme : **Nom\_variable type**
- Il est possible de déclarer une variable par référence à une colonne d'une table par la notation **%type**, selon la syntaxe :

**Nom\_variable nom\_table.nom\_colonne%type**

- **Types composés** : PL/SQL offre deux types composés : enregistrement (RECORD) et table (TABLE) :

- **a. Enregistrement** : la déclaration d'une variable de ce type se fait :
- Soit par référence à une structure de table ou de curseur, en utilisant la notation **%ROWTYPE** :

```
nom_variable nom_table%ROWTYPE;  
nom_variable nom_curseur%ROWTYPE;
```

- Soit par énumération des rubriques qui la composent :

Dans le second cas, la déclaration s'effectue en deux étapes :

### 1. Déclaration du type enregistrement :

```
TYPE NOM_TYPE IS RECORD  
(nom_champ type champ,...);
```

### 2. Déclaration de la variable de type enregistrement :

```
nom_variable NOM_TYPE;
```

Exemple : déclaration de la variable rev\_employe :

```
TYPE Revenu IS RECORD (  
    Nom Employe.nom%type,  
    Salaire NUMBER (9,2)  
);  
rev_employe Revenu ;
```

- **b. Table** : une structure composée d'éléments d'un même type scalaire.

L'accès à un élément de la table s'effectue grâce à un indice, ou clé primaire, déclaré de type **BINARY\_INTEGER** qui permet de stocker des valeurs entières signées.

### 1. Déclaration du type de l'élément de la table :

```
TYPE NOM_TYPE IS TABLE OF type_champ
INDEX BY BINARY_INTEGER;
```

### 2. Déclaration de la variable de type table :

```
Nom_variable NOM_TYPE ;
```

Exemple : déclaration de la variable t\_nom :

```
TYPE table_nom IS TABLE OF VARCHAR2 (35)
INDEX BY BINARY_INTEGER;
t_nom table_nom ;
```

**Valeur initiale** : il est possible d'attribuer une valeur initiale à une variable au moment de sa déclaration :

```
Nom_variable type :=valeur ;
```

**Constante** : la définition d'une constante :

```
Nom_variable type DEFAULT valeur ;
```

Ou : Nom\_variable CONSTANT type :=valeur ;

## 1.2.2 Visibilité d'une variable

La variable est utilisable dans le bloc où elle est définie ainsi que dans les blocs imbriqués dans le bloc de définition.

### 1.2.3 Conversion de types

Les conversions de types de données peuvent être :

- Implicites, par conversion automatique.
- Explicite, par utilisation de fonctions SQL telles que TO\_DATE, TO\_CHAR, TO\_NUMBER ;

Exemple : TO\_CHAR(1234,'09999999') = 0001234

Quelques formats de conversion de date : **TO\_CHAR**(date[, 'format'])

FORMAT étant la combinaison de codes suivants :

YYYY	Année
YY	2 derniers chiffres de l'année
MM	numéro du mois
DD	numéro du jour dans le mois
HH	heure sur 12 heures
HH24	heure sur 24 heures
MI	minutes
SS	secondes

Exemple :

```
SELECT  
    TO_CHAR(SYSDATE,'DD MM YYYY HH24: MI')  
FROM dual ;
```

==> 01 10 2004 09 : 24

## 1.3 Les instructions

### 1.3.1 Les instructions d'affectations

Il existe trois moyens d'affecter une valeur à une variable :

- l'opérateur d'affectation
- L'ordre FETCH (section sur les curseurs)
- L'option INTO de l'ordre SELECT

#### a/ Opérateur d'affectation :

- Variable de type simple : Nom\_variable : = valeur ;
- Variable de type composé :
  - Variable de type table : nom\_variable (valeur clé primaire):=valeur ;

#### Exemple :

```
DECLARE
TYPE Table_nom IS TABLE OF CHAR (35)
INDEX BY BINARY_INTEGER;
t_nom Table_nom ;
i BINARY_INTEGER;
```

L'affectation de la chaîne de caractères 'Info' au deuxième poste de table\_nom peut se faire par :

t\_nom (2) := 'Info' ;

Ou par : i := 2 ; t\_nom(i) := 'Info' ;

- Variable de type enregistrement : nom\_variable.nom\_champ ;

Exemple :

```
DECLARE
TYPE t_employe IS RECORD
    (nom_employe   Employe.nom%otype,
    revenu_employe number(8, 2)
    );
emp   t_employe ;
```

L'affectation d'un revenu de 12 000,00 à l'employé de nom Alaoui s'effectue par :

```
emp.nom_employe := 'Alaoui' ;
emp.revenu_employe :=12000.00 ;
```

### **b/ Valeur résultat d'une requête :**

L'utilisation de la clause INTO de l'ordre SELECT permet d'affecter à une variable le résultat d'une requête.

SELECT liste d'expressions

INTO liste de variables

FROM...

Exemple :

```
DECLARE
    u_nom Employe.nom%otype ;
    u_sal Employe.sal%otype;
BEGIN
    SELECT nom, sal INTO u_nom, u_sal
    FROM Employe WHERE Numero = 7000;
END;
```



En cas d'enregistrement :

```
SELECT liste d'expressions  
INTO nom d'enregistrement  
FROM...
```

Exemple:

```
DECLARE  
TYPE t_emprec IS RECORD  
    (r_nom Employee.nom%type,  
     r_sal Employee.sal%type);  
emprec t_emprec;  
BEGIN  
    SELECT nom, sal    INTO emprec  
    FROM Employee  
    WHERE numero = 7000;  
END;  
/
```

### 1.3.2 Les instructions de contrôle

a/ **Structure alternative** : exécution d'instructions sous le contrôle d'une condition.

```
IF condition THEN  
    Instructions ;  
[ELSIF condition THEN  
    instructions; ]  
[ELSE  
    instructions ;]  
END IF ;
```

b/ **Structures répétitives** :

```
Boucle infinie : LOOP  
    instructions ;  
END LOOP ;
```

Boucle Pour : FOR var\_indice IN [REVERSE] valeur\_début ... valeur\_fin

```
LOOP
    instruction ;
END LOOP ;
```

Boucle tant que :

```
WHILE condition
LOOP
    Instructions ;
EXIT WHEN condition ; uniquement autorisé pour sortir d'une
boucle infinie.
END LOOP;
```

**Exemple :**

```
DECLARE
    i NUMBER(3);
BEGIN
    FOR IN 1..100
    LOOP
        INSERT INTO Employe (noemp, nomemp, job, nodept)
            VALUES (i,'XX','Developpeur', 1);
    END LOOP;
END;
/
```

## 1.4 Curseur

Le curseur est une structure de données permettant de stocker le résultat d'une requête qui retourne **plusieurs tuples** (lignes).

Il s'agit de zones de mémoire allouées spécifiquement pour traiter les instructions SQL.

### Deux types de curseurs:

- implicite : créés automatiquement par Oracle pour ses propres traitements.
- explicite: créés par l'utilisateur pour pouvoir traiter le résultat de requêtes retournant plus d'un tuple.

### 1.4.1 Utilisation du curseur

#### a. Déclaration

Déclaration : **CURSOR** nom\_curseur **IS** requête ;

Un curseur peut aussi être défini à l'aide de paramètres :

```
CURSOR nom_curseur (  
    Nom_paramètre type [ :=valeur par défaut] [...])  
IS requête;
```

#### Exemple :

```
DECLARE  
CURSOR C1 IS SELECT nom FROM Employe WHERE sal = 1000;  
CURSOR C2 ( Psal NUMBER (7, 2), pcom NUMBER (7, 2))  
IS SELECT ename FROM EMP WHERE sal=psal AND comm=pcom;
```

**N.B** : un tuple du curseur sera de type C2%ROWTYPE.

## b. Ouverture du curseur

L'ordre OPEN permet d'allouer un espace mémoire au curseur :

```
OPEN nom_curseur ;
```

Ou : OPEN nom\_curseur (paramètres effectifs) ;

### Exemple :

```
OPEN C1 ;  
OPEN C2 (12000, 2500) ;
```

## c. Fermeture du curseur

L'ordre CLOSE libère la place mémoire : CLOSE nom\_curseur ;

## d. Traitement des lignes

Pour récupérer les tuples successifs de la requête, on utilise l'instruction :

```
FETCH nom_curseur INTO liste_variables ;
```

### Exemple1 :

```
DECLARE  
  Type t_emp IS RECORD  
    (V_nom Employee.nom%type,  
     v_sal Employee.sal%type);  
  R_emp t_emp;  
  CURSOR C3 IS SELECT nom, sal FROM Employee;  
BEGIN  
  OPEN C3;  
  LOOP  
    FETCH C3 INTO r_emp;  
    EXIT WHEN (C3%NOTFOUND);  
    ....  
  END LOOP;  
  CLOSE C3;  
END;  
/
```

Lorsqu'on souhaite parcourir un curseur dans une boucle pour effectuer un traitement, on peut simplifier l'utilisation de ceux-ci.

```
DECLARE
CURSOR EmpCur IS SELECT * FROM Employe WHERE
SAL<=6000.00;
emp Employe%ROWTYPE;
BEGIN
  Open EmpCur;
  FOR emp IN EmpCur
  LOOP
    UPDATE Employe
    SET SAL = SAL + 500.00
    WHERE numero=emp.numero;
  END LOOP;
END;
/
```

L'incrémentation du curseur EmpCur est implicite.

### 1.4.2 Attributs des curseurs

%NOTFOUND est égal à False si FETCH retourne un résultat

%FOUND est l'opposé logique de %NOTFOUND

%ROWCOUNT renvoie le nombre de lignes lues

%ISOPEN est égal true si le curseur est ouvert

## 1.5 La gestion des exceptions

Il s'agit d'affecter un traitement approprié aux erreurs qui apparaissent lors de l'exécution du bloc PL/SQL.

Un certain nombre d'exceptions sont prédéfinies sous Oracle. Citons, pour les plus fréquentes : NO\_DATA\_FOUND (devient vrai dès qu'une requête renvoie un résultat vide),

CURSOR ALREADY OPEN (curseur déjà ouvert), INVALID CURSOR (curseur invalide)...

L'utilisateur peut définir ses propres exceptions.

Déclaration (section déclaration) :

Nom_exception EXCEPTION ;		
<u>Exemple</u> :	excpt1	EXCEPTION
El Qadi	-75-	SQL & PL/SQL Oracle

Déclenchement l'exception (section BEGIN) :

```
IF condition THEN
    RAISE nom_exception
END IF
```

Traitement de l'exception (section EXCEPTION) :

```
EXCEPTION
    WHEN <exception1> [OR <exception2> OR ...]
    THEN <instructions>
    WHEN <exception3> [OR <exception2> OR ...]
    THEN <instructions>
    WHEN OTHERS THEN <instructions>
END;
```

### **Exemple :**

```
CREATE TABLE erreur (lib1 char (15), lib2 char (50));
DECLARE
    Erreur_comm EXCEPTION;
    V_nom Employe.nom%type;
    V_sal Employe.sal%type;
    v_comm Employe.comm%type;
BEGIN
    SELECT nom, sal, comm INTO v_nom, v_sal, v_comm FROM Employe
    WHERE numero=&num_emp;
    IF v_sal = v_comm THEN RAISE erreur_comm;
    End if;
        INSERT INTO erreur VALUES (v_nom, 'OK');
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO erreur VALUES ('Employé inconnu', NULL);
    WHEN erreur_comm THEN
        INSERT INTO erreur VALUES (V_nom, 'comm > sal');
END;
/
```

El Qadi

-77-

SQL & PL/SQL Oracle

SELECT \* FROM erreur;

Cette procédure PL/SQL envoie dans la table erreur, le message 'nom\_Employé ok' si le numéro de l'employé existe, sinon le message 'Employé inconnu'. De plus, si sal < comm, elle envoie dans la table erreur le message 'nom\_Employé comm>sal'.

El Qadi

-78-

SQL & PL/SQL Oracle

## 2. Procédures et fonctions stockées

Il est possible de créer des procédures et des fonctions comme dans n'importe quel langage de programmation classique.

La syntaxe de création d'une procédure est la suivante :

```
CREATE OR REPLACE PROCEDURE <NOM_PROC>
[(Liste de paramètres)]
AS <zone de déclaration de variables>
BEGIN
    <Corps de la procédure>
EXCEPTION
    <traitement des exceptions>
END;
```

On peut aussi créer des fonctions. Le principe est le même, l'en-tête devient :

```
CREATE OR REPLACE FUNCTION <NOM_FONCTION>
[(Paramètres)]
RETURN <type du résultat>
```

A noter que toute fonction ou procédure créée devient un objet part entière de la base (comme une table ou une vue, par exemple). Elle est souvent appelée “procédure stockée”.



## 2.1 Les paramètres

Toute procédure ou fonction doit être définie avant d’être appelé

- Définition dans la section déclaration d’un bloc PL/SQL.

Il y a trois façons de passer les paramètres dans une procédure ou une fonction :

- IN (lecture seule) : paramètres qui ne doivent pas être modifiés par sous-programme.
- OUT (écriture seule) : pour les paramètres transmis en résultat.
- INOUT (lecture et écriture) : pour les variables dont la valeur peut être modifiée en sortie et consulté par la procédure

## 2.2 Appel aux procédures et aux fonctions :

L’appel à une procédure stockée se fait par la commande :

**SQL>EXECUTE nom\_procedure (liste de paramètres effectifs) ;**

L’appel à une fonction stockée se fait par la commande :

**SQL>EXECUTE : variable\_locale : = nom\_fonction  
(liste de paramètres effectifs) ;**

### Exemple1 :

```
CREATE TABLE T (num1 INTEGER, num2 INTEGER);
CREATE PROCEDUREessai (x IN NUMBER (4), y OUT NUMBER (4),
z INOUT NUMBER (4))
IS
BEGIN
    y:=x*z;
    z:=z*z;
    INSERT INTO T VALUES(y, z);
END essai;
/

DECLARE
    a NUMBER(4);
    b NUMBER(4);
BEGIN // appel de la procédure à partir du bloc PL/SQL
    a:=5;
    b:=10;
    essai(2,a,b);
    a:=a*b;
END;
/
```

Après l'appel, le couple (a, b) vaut (20, 100)  
et c'est aussi le tuple qui est inséré dans T.

### Exemple2:

Calcul le nombre d'employés affectés à un département :

```
CREATE OR REPLACE FUNCTION Nbre_Emp_Dept
(x_code IN DEPT.deptno%type) RETURN NUMBER
IS
    NbEmpl    NUMBER:=0;
BEGIN
    SELECT Count (Empno) INTO nbEmpl
    FROM emp
    WHERE deptno = x_code;
    RETURN (nbEmpl);
END Nbre_Emp_Dept;
/
```

// Appel de la fonction en mode interactif

```
declare
  nbre NUMBER:=0;
begin
  nbre:=Nbre_Emp_Dept (&numdept) ;
  dbms_out.put_line(nbre);
end;
/
```

## 2.3 Suppression d'une procédure et d'une fonction :

DROP PROCEDURE <nom\_procedure>;

DROP FUNCTION <nom\_fonction>;

## 3. Les déclencheurs

– On appelle déclencheur, ou trigger, un traitement déclenché par un événement.

Par exemple, un déclencheur peut être défini pour vérifier, lors de chaque affectation d'un employé à un département que l'employé n'est pas déjà affecté à un autre département.

Un trigger sera un objet stocké (comme une table ou une procédure).

Un déclencheur de BD est associé à une seule table ; il est opérationnel jusqu'à la suppression de la table à laquelle il est lié.

### 3.1 Création des déclencheurs

```
CREATE [OR REPLACE] TRIGGER <nom>
{BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <nom de table>
[FOR EACH ROW [WHEN (<condition>)]]
<Corps du trigger>
```

- Un trigger se déclenche avant ou après (BEFORE|AFTER) une insertion, destruction ou mise à jour (INSERT|DELETE|UPDATE) sur une table (à noter que l'on peut exprimer des conditions plus complexes avec le OR: INSERT OR DELETE ...).
- L'option FOR EACH ROW [WHEN (<condition>)] fait s'exécuter le trigger à chaque modification d'une ligne de la table spécifiée (on dit que le trigger est de "niveau ligne"). En l'absence de cette option, le trigger est exécuté une seule fois ("niveau table").

#### Exemple1:

```
CREATE TABLE T1 (num1 INTEGER, num2 INTEGER);
CREATE TABLE T2 (num3 INTEGER, num4 INTEGER);

CREATE TRIGGER inverse
AFTER INSERT ON T1
FOR EACH ROW WHEN (NEW.num1 <=3)
BEGIN
    INSERT INTO T2 VALUES (:NEW.num2, :NEW.num1);
END inverse;
/
```

Ce trigger va, en cas d'insertion d'un tuple dans T1 dont la première coordonnée est inférieure ou égale à 3, insérer le tuple inverse dans T2.

Les préfixes NEW et OLD (en cas de UPDATE ou de DELETE) vont permettre de faire référence aux valeurs des colonnes après et avant les modifications dans la table.

Ils sont utilisés sous la forme NEW.num1 dans la condition du trigger et sous la forme : NEW.num1 dans le corps.

## 3.2 Activation et désactivation des déclencheurs

```
ALTER TRIGGER <nom_trigger> {ENABLE|DISABLE};
```

Activer et désactiver les déclencheurs associés à une même table.

```
ALTER TABLE <nom_table> {ENABLE|DISABLE} ALL TRIGGERS;
```

## 3.3 Suppression des déclencheurs

```
DROP TRIGGER <nom_trigger>;
```

## 4. Les packages

Un package est un regroupement de procédures et de fonctions.

Une telle unité se divise en deux parties distinctes :

- le corps qui contient l'ensemble de toutes les définitions de procédures et de fonctions,
- et l'interface (partie spécification) qui spécifie celles d'entre elles qui sont utilisables de l'extérieur du package.

## 4.1 Création de package

### a) création de la partie spécification

```
CREATE [OR REPLACE] PACKAGE nom_package  
[IS | AS]  
    { Déclaration de procédure ;  
      .....  
    }  
END nom_package ;  
/
```

### b) création de la partie corps (BODY)

```
CREATE [OR REPLACE] PACKAGE BODY nom_package  
[IS | AS]  
    {[Déclaration de procédure]  
    | [Déclaration de fonction]  
    .....}  
END nom_package ;  
/
```

## 4.2 Utilisation de package

L'appel à un élément du package se fait en préfixant son nom par le nom du package.

```
declare  
begin  
    nom_package.nom_procedure  
end;
```

## 4.3 Modification d'un package existant

Pour modifier la partie spécification ou la partie corps d'un package, il suffit de modifier le texte source correspondant et d'exécuter l'un des ordres :

```
REPLACE PACKAGE nom_package ;  
REPLACE PACKAGE BODY nom_package ;
```

## 4.4 Suppression d'un package

```
DROP PACKAGE nom_package ; // suppression la totalité du package  
DROP PACKAGE BODY nom_package ; // suppression la partie corps du package
```

## 4.5 Package DBMS\_OUTPUT

**Affichage :** PL/SQL n'est pas un langage avec des fonctionnalités d'entrées sorties évoluées. Toutefois, on peut imprimer des messages et des valeurs de variables de plusieurs manières différentes. Le plus pratique est de faire appel à un package prédéfini : **DBMS\_OUTPUT**.

En mode interactif, il faut exécuter la commande :

**SET SERVEROUTPUT ON** avant le code à tester.

Pour afficher, on utilise la commande suivante :

**DBMS\_OUTPUT.PUT\_LINE('Au revoir' || nom || ' a bientôt');**

### Debugger

Pour chaque objet créé (procédure, fonction, trigger...), en cas d'erreur de compilation, on peut voir ces erreurs en tapant (si "essai" est un nom de procédure)

**SHOW ERRORS PROCEDURE** essai;

**Atelier 4.** On souhaite gérer les résultats d'examens du semestre S3 de deux filières "F1" et "F2". Il s'agit de définir des programme PL/SQL permettant l'insertion automatique d'informations dans les deux relations RESULTAT et CLASSEMENT, à partir des données des relations NOTATION et MATIERE, qui contiennent respectivement des renseignements sur les notes obtenues par les étudiants et les coefficients affectés aux matières.

- Etudiant (numero char(4) primary key, nom varchar(10), codefil char(2));
  - Matiere (codemat char (2) primary key, CE number(4,2), CP num|ber(4,2));
  - Notation (numero char(4), codemat char(2), NE number(7,2), NP number(7,2));
- Exemple de jeux de données des trois tables.

Etudiant (120 Etudiants)		
numero	nom	codefil
E1	ETUDI1	F1
E2	ETUD2	F2
....		
E120	ETUDI20	F1

Matiere (4 Matières)		
codemat	CE	CP
M1	0.5	0.5
M2	0.8	0.2
M3	0.7	0.3
M4	0.6	0.4

Notation:				
(480 Notes)				
E1,M1,1,5,12	E1,M2,16,8	E1,M3,12,13	E1,M4,16,17	
E2,M1,1,2,8	E2,M2,NULL,13	E2,M3,7,9	E2,M4,17,5,NULL	
E30,M1,7,5	E3,M2,8,9	E3,M3,NULL,12	E3,M4,NULL,NULL	
...	...	...	...	
E120,M1,12,12	E120,M2,13,8	E120,M3,10,13	E120,M4,15,17	

- 1) Créer sous votre compte les trois tables ;
- 2) Créer un bloc PL/SQL permettant d'initialiser les trois relations avec un nombre conséquent de tuples (de l'ordre de 120 étudiants, de 4 matières, et de 480 notations). Utiliser le package dbms\_random.value(valeur\_debut,valeur\_fin) pour insérer des valeurs aléatoires dans les champs : CE, CP, NE, NP, et codefil. la valeur NULL remplace la valeur -1.
- 3) Créer la procédure "Détail\_Filière (codefilière)", qui affiche la liste des étudiants inscrits dans cette filière. Le message d'erreur (ORA-20010, "Code Filière n'existe pas") sera émis si le codefilière n'existe pas dans la table Etudiant. Afficher le numéro et le nom d'étudiant, en utilisant la package DBMS\_OUTPUT.PUT\_LINE.
- 4) Créer sous votre compte la table: RESULTAT  
RESULTAT (numero char(4),nom varchar(10), codemat char(2), NG number(7,2));  
NG représente la note globale de la matière en question.

EI Qadi

-95-

SQL & PL/SQL Oracle

- 5) Créer une procédure stockée nommée RESULTAT\_ETUDIANT qui insère dans la relation RESULTAT tous les tuples constitués du numéro d'un étudiant, de son nom, et de la note globale de la matière en question ; NG = (NE × CE) + (NP × CP)  
Si la note écrit est NULL (ou la note pratique est NULL) alors NG = NULL
- 6) Créer sous votre compte la table: CLASSEMENT  
CLASSEMENT (numero char(4),nom varchar(10), moygen number(7,2),rang integer);  
rang : représente classement de l'étudiant en fonction de sa moyenne générale.
- 7) Créer une procédure stockée nommée CLASSEMENT\_ETUDIANT qui insère dans la relation CLASSEMENT tous les tuples constitués du numéro d'étudiant, de son nom, la moyenne générale obtenue dans toutes les matières par cet étudiant; et le rang (place) doit être calculé.  
Si NG est NULL alors moygen = NULL et rang=NULL
- 8) Ecrire la procédure RESULTAT\_FILIERE (codefilière) qui donne le nombre total des étudiants, le nombre des étudiants ayant validés semestre, le nombre des étudiants non validés, et le nombre des étudiants ayant la moyenne générale égale à NULL dans la filière de codefil=codefilière. La validation est conditionnée par une note>=12.
- 9) Créer une procédure stockée nommée POINTJURY (codeetud, codemat, notejury) qui modifie la note globale d'une matière et la moyenne générale d'un étudiant selon la règle suivante :
  - nouvelle note globale = ancienne note + notejury;
  - nouvelle moyenne générale = moyenne des 4 notes globales.
- 10)Créer une fonction stockée nommée GET\_RESULTAT\_MATIERE (codematiere) afin d'extraire le nombre total des étudiants ayant validé cette matière (note globale ≥ 12). Un message d'erreur sera généré si la valeur du paramètre d'entrée n'existe pas.
- 11)Ecrire la fonction POURC\_FILIERE (codefilière) qui donne le pourcentage des étudiants ayant validés semestre par rapport au nombre total des étudiants inscrits dans cette filière.
- 12)Créer un déclencheur qui interdisse toute insertion d'une ligne dans la table MATIERE.
- 13)Créer un déclencheur qui permette de suivre toutes les modifications et suppressions de lignes dans la table NOTATION. En cas de modification, on notera le numéro d'étudiant, le code matière, la date et l'heure de la modification et le nom de chaque colonne concernée. En cas de suppression, on notera le numéro d'étudiant, le code de la matière, la date et l'heure de suppression.
- 14)Créer un trigger qui empêche la modification du numéro d'étudiant dans la table Etudiant.
- 15)Créer un trigger qui permet d'initialiser la colonne 'NE' à NULL, et 'NP' à NULL dans la table NOTATION.
- 16)Créer une spécification et un corps de package nommé ETUDIANT\_PKG contenant les fonctions GET\_RESULTAT\_MATIERE, POURC\_FILIERE, et la procédure RESULTAT\_FILIERE.

EI Qadi

-96-

SQL & PL/SQL Oracle