Les bases de données informatiques sont utilisées dans un grand nombre d'entreprises pour stocker, organiser et analyser les données. Découvrez tout ce que vous devez savoir à ce sujet : qu'est-ce qu'une base de données, à quoi sert-elle, comment fonctionne-t-elle etc.

Une base de données est une **collection d'informations organisées afin d'être facilement consultables, gérables et mises à jour**. Au sein d'une base, les données sont organisées en lignes, colonnes et tables. Elles sont indexées afin de pouvoir facilement trouver les informations recherchées à l'aide d'un logiciel informatique. Chaque fois que de nouvelles informations sont ajoutées, les données sont mises à jour, et éventuellement supprimées.

Qu'est-ce qu'une base de données ?

Une BdD (entendez Base de Données) se charge elle-même de créer, de mettre à jour ou de supprimer des données. Elle **effectue également des recherches parmi les données qu'elle contient sur demande de l'utilisateur**, et de lancer des applications à partir des données.
Les bases de données sont utilisées par la presque totalité des entreprises à travers le monde. Elles sont notamment utilisées par les universités, les compagnies aériennes pour gérer les réservations, les hôpitaux et cliniques pour la gestion du personnel et des patients, etc. Les bases de données les plus larges sont généralement utilisées **par les agences gouvernementales**, les grandes entreprises ou les universités.

Comment fonctionnent les bases de données ?

Les bases de données sont stockées sous forme de fichiers ou d'ensemble de fichiers sur un disque magnétique, un disque optique ou tout autre type de stockage. Les bases de données traditionnelles (hiérarchiques) sont organisées par **champs (fields), enregistrements et fichiers**. Un champ est une seule pièce d'information. Un enregistrement est un ensemble de champs. Un fichier est une collection d'enregistrements.
Par exemple, une **fiche de notes est l'équivalent d'un fichier**. Elle contient un ensemble d'enregistrements, et chaque enregistrement regroupe plusieurs champs : matricule, nom, prénom, moyenne etc. En guise d'exemple, on peut aussi citer les catalogues de produits ou les inventaires.
La faculté de consulter ou de modifier une BdD (lire ou écrire) est conférée aux divers utilisateurs par un **DBA (Data Base Administrator)**.

SGBD (système de gestion de base de données) ou DBMS et RDBMS : les logiciels qui permettent d'accéder aux bases de données

Pour accéder aux bases de données, on utilise un SGBD (système de gestion de base de données), à savoir un **logiciel de type DBMS ou RDBMS**. Un DBMS est un database management system (système de gestion de database). Il permet de définir, manipuler, récupérer et gérer les données stockées au sein de la BdD. Le DBMS extrait des informations de la BdD sur demande de l'utilisateur, en fonction des requêtes effectuées. Pour effectuer une requête, l'utilisateur peut entrer un mot-clé, ou effectuer une commande de tri.
La puissance d'un DBMS est sa capacité à **définir de nouvelles relations** à partir de relations basiques données par les tables pour répondre aux requêtes. Généralement, l'utilisateur entre une série de caractères, et l'ordinateur cherche les séquences correspondantes pour fournir à l'utilisateur les matériaux sources dans lesquels ces caractères apparaissent. Par exemple, un utilisateur peut rechercher tous les enregistrements contenant un champ lié à une personne portant le nom de famille DIALLO.

Qu'est-ce que le SQL ou Structured Query Language ?

Le SQL (Structured Query Language) est un **langage de programmation standardisé** utilisé pour gérer les bases de données relationnelles et effectuer différentes opérations sur les données qu'elles contiennent. Ce langage a été créé dans les années 70, et continue d'être utilisé régulièrement par les

administrateurs de bases de données. Les développeurs qui rédigent des scripts d'intégration de données et les analystes de données l'utilisent aussi pour lancer des requêtes analytiques.
Un standard SQL officiel a été adopté par l'ANSI (Americain National Standards Institute) en 1986. En 1987, l'ISO (International Organization for Standardization) l'a adopté à son tour. Ce standard a été mis à jour plus de six fois depuis lors. La **version la plus récente est le SQL:2011**.
L'utilisation du SQL permet d**e modifier les structures des tables et des index des bases de données**. L'utilisateur peut ajouter, mettre à jour et supprimer des rangées de données et retrouver des sous-ensembles d'informations. Ces informations peuvent ensuite être utilisées pour les applications analytiques ou les traitements de transactions. Une r**equête SQL prend la forme d'une commande écrite**. Les commandes les plus fréquemment utilisées sont la sélection, l'ajout, l'insertion, la mise à jour, la suppression, la création, l'altération et le tronquement.

Commandes SQL et les différents outils

Les **commandes SQL sont divisées en plusieurs types différents**. Le vocabulaire LMD (langage de manipulation des données) permet de retrouver et de manipuler des données. Le LDD (langage de définition des données) est utilisé pour définir et modifier les structures des bases de données. Le contrôle de transactions permet de gérer les transactions pour s'assurer qu'elles soient complétées ou annulées en cas de problème ou d'erreur. Enfin, les security statements sont utilisés pour contrôler l'accès aux bases de données et créer un système de permissions pour les différents utilisateurs.
Les entreprises utilisent des RDBS ou systèmes de gestion de base de données développés autour de SQL propriétaires ou open source. Parmi les plus connus, on compte Microsoft SQL Server, Oracle Database, IBM DB2, SAP HANA, SAP Adaptive Server, Oracle MySQL et PostgreSQL. Malgré la standardisation du SQL, la plupart des vendeurs utilisent des **extensions propriétaires** pour la programmation procédurale et les autres fonctions. Par exemple, Microsoft propose un set d'extensions, le Transact-SQL (T-SQL) et Oracle a sa version étendue (PL/SQL). De fait, les variantes des divers vendeurs ne sont pas compatibles entre elles.

Qu'est-ce qu'un administrateur de base de données (Data Base Administrator) ?

Un Administrateur de base de données ou Data Base Administrator est la **personne chargée de maintenir un environnement de ce type**. La conception, l'implémentation, la maintenance du système et la mise en place de règles. Il doit aussi former les employés de l'entreprise à la gestion et à l'utilisation de la BdD.
En règle générale, un **DBA dispose d'une formation dans le domaine des sciences informatiques** et d'une expérience professionnelle avec une spécifique ou diverses bases de données. Il doit aussi avoir une expérience avec les principaux produits de gestion de database comme SQL, SAP ou Oracle.

Commandes de base

Comme pour le langage SQL, les commandes de SQL*Plus sont insensibles à la casse.

Le tableau suivant indique les commandes de base pour manipuler le buffer d'entrée de toutes les interfaces, sauf pour iSQL*Plus qui propose ces options d'une manière intuitive via des boutons.

Commandes	Commentaires
R	Exécute (*run*).
L	Liste le contenu du buffer.
L*	Liste la ligne courante.
L*n*	Liste la *nième* ligne du buffer qui devient la ligne courante.
L *n m*	Liste les lignes de *n* à *m*
n	Indique quelle doit être la ligne courante
n *text*	Remplace la ligne par *text*
I	Insère une ligne après la ligne courante.
A *texte*	Ajoute *texte* à la fin de la ligne courante.
DEL	Supprime la ligne courante.
C/*texte1*/*texte2*/	Substitution de la première occurrence de *texte1* par *texte2* dans la ligne courante.

CLEAR	Efface le contenu du buffer.
QUIT ou EXIT	Quitte SQL*Plus.
CONNECT *user/ Password@descripteur*	Autre mode de connexion (sans sortir de l'interface).
GET *fichier*	Charge dans le buffer le contenu du *fichier.sql* qui se trouve dans le répertoire courant.
SAVE *fichier*	Écrit le contenu du buffer dans *fichier.sql* qui se trouve dans le répertoire courant.
START *fichier* ou *@fichier*	Charge dans le buffer et exécute *fichier.sql*.
SPOOL *fichier*	Crée *fichier.lst* dans le répertoire courant qui va contenir la trace des entrées/sorties jusqu'à la commande SPOOL OFF

Tables utilisées dans le cours :

emps

Numemp	Titre	Nom	Prenom	Fonction	Refsup	Datemb	Sal	Prime	Numdept
4834	Dr.	Barry	Faza	Recteur		02/01/02	5500		1
4536	M.	Camara	Balla	Chfdpt	4834	04/01/03	3650		1
4957	M.	Camara	Mamadi	Chfdpt	4834	03/07/08	3500		3
4852	M.	Cisse	Sidiki	Chfdpt	4834	10/03/03	3400		4
4136	M.	Dioubate	Fantamadi	Chfdpt	4834	12/06/06	3650		2
4439	M.	Sow	Nouhou	Informaticien	4536	04/02/03	3390		1
4224	Mme	Sylla	Nana	Informaticien	4536	08/03/08	2500		1
4114	M.	Kaba	Amadou	Economiste	4957	13/08/12	1700	1750	3
4220	M.	Diallo	Oumar	Economiste	4957	18/04/12	2100	800	3
4022	Mme	Barry	Aissatou	Informaticien	4536	24/01/03	2600		1
4330	Mlle	Diaby	Adama	Magistrat	4136	21/02/12	3350		2
4350	M.	Barry	Hamidou	Magistrat	4136	27/04/12	3250		2
4035	M.	Fofana	Ousmane	Magistrat	4136	20/05/08	1200		2
4530	Mme	Balde	Aminata	Informaticien	4536	12/03/15	1350		1
4110	M.	Kamano	Aly	Economiste	4957	10/08/05	1450	0	3
4011	Mlle	Soumah	Kadiatou	Magistrat	4136	06/02/16	1500		2
4730	M.	Kante	Bano	Economiste	4957	03/10/03	3150	150	3
4545	M.	Camara	Alpha	Sociologue	4852	01/01/15	3200		4
4121	Mme	Bah	Fatou	Sociologue	4852	06/03/16	1450		4
4211	M.	Diallo	Talibe	Informaticien	4536	04/04/11	3120		1
4428	Mme	Sidibe	Macire	Sociologue	4852	04/04/08	3100		4
4129	M.	Cisse	Naby	Informaticien	4530	05/02/16	1300		1
4131	Mlle	Barry	Issa	Comptable	4439	07/03/16	1325		1

depts

Numdept	Nomdept	Zone
1	Informatique	Lambanyi
2	Droit	Kipece
3	Economie	Kipedia
4	Sociologie	Kaporo

gradesal

Grade	Salmin	Salmax
1	800	1400
2	1401	1600
3	1601	2200
4	2201	3100
5	3101	6000

1. Sélection et tri des lignes retournées par un ordre SELECT

1.1. Objectifs :

Au terme de ce paragraphe, l'apprenant doit être capable de

- Limiter le nombre de lignes retournées par une requête
- Trier les lignes retournées par une requête

1.2. Sélectionner des lignes

La clause WHERE permet de spécifier une condition à satisfaire et se place immédiatement après la clause FROM. Elle compare des valeurs dans des colonnes, des littéraux, des expressions arithmétiques ou des fonctions. Elle comprend 3 éléments :

- Nom de la colonne
- Opérateur de comparaison
- Nom de colonne, constante ou liste des valeurs

Exemple :

```
SQL> Select nom, fonction, numdept
        from emps
        where fonction = 'Informaticien';
```

N.B: la fonction Informaticien a été saisie avec l'initial en majuscule pour garantir la correspondance avec la colonne fonction de la table emps. *En effet, la recherche tient compte des minuscules et des majuscules.*

Dans la clause WHERE, les chaines alphanumériques et les dates doivent être inclues entre simples quotes (' '), mais pas les constantes numériques.

Le format de date par défaut est jj-mmm-aa

1.3. Opérateurs de comparaison

= égal à

> supérieur à

≥ supérieur ou égal à

≤ inférieur ou égal à

< inférieur à

< > différent de

Ces opérateurs s'utilisent dans les conditions qui comparent deux expressions. Dans la clause WHERE, ils s'utilisent de la manière suivante :

…WHERE expression opérateur valeur

Exemples :

WHERE datemb = '04/01/86'

WHERE sal <= 2500

WHERE nom = 'Barry'

```
SQL> select nom, prénom, sal
        from emps
        where sal >= 2500;

SQL> select nom, prénom, sal
        from emps
        where sal – 1300 = prime;
```

```
SQL> select nom, prénom, sal, prime
        from emps
        where sal<= prime;
```

Dans ce dernier exemple, l'ordre select recherche le nom, le prénom, le salaire et la prime, la condition étant que le salaire de l'employé soit inférieur ou égal à la prime. Noter ici qu'aucune valeur explicite n'est fournie dans la clause WHERE. Les 2 valeurs à comparer sont prises dans les colonnes SAL et PRIME de la table *emps*.

1.4. Autres opérateurs de comparaison

Exemples :

- L'ordre select ci-dessous ramène les lignes correspondant aux employés ayant un salaire compris entre 2000 et 3000

```
SQL>Select nom, prénom, sal
        from emps
        where sal between 2000 and 3000;
```

Pour comparer une expression avec une liste de valeurs, utiliser l'opérateur IN. Afficher le numéro d'employé, le nom, le prénom, le salaire et la référence du supérieur (Refsup) des employés ayant un supérieur dont le matricule (numemp) est 4834, 4957 ou 4852.

```
SQL>Select numemp nom, prénom, sal
        from emps
        where refsup in (4834, 4957, 4852);
```

- Lorsque vous ignorez les valeurs exactes à rechercher, vous pouvez sélectionner des lignes correspondant à une suite de caractères au moyen de l'opérateur LIKE. L'opération ainsi exécutée est appelée une recherche générique. Deux symboles sont utilisés pour la recherche :
  - % représente zéro caractère ou une séquence quelconque de caractères
  - _ représente un caractère quelconque

Exemple :

L'ordre SELECT ci-dessous ramène tous les employés dont le nom commence par « C »

```
SQL> Select nom
        from emps
        where nom like 'C';
```

Il s'agit bien de "C" majuscule. Des noms commençant par « c » (s'il en existe) ne seront pas sélectionnés.

1.5. L'opérateur IS NULL

Il teste les valeurs Null. Une valeur null est indisponible, non attribuée, inconnue ou inapplicable. Il est donc impossible de tester une valeur null à l'aide de l'opérateur =, puisqu'elle ne correspond à aucune condition d'égalité ou d'inégalité. L'exemple ci-dessous recherche le prénom et le supérieur hiérarchique (Refsup) de tous les employés n'ayant pas de supérieur.

```
SQL> select prenom, refsup
        from emps
        where refsup is null;
```

Autre exemple : afficher le prénom, la fonction et la prime de tous les employés ne bénéficiant pas de prime.

```
SQL> select prenom, fonction, prime
        from emps
        where prime is null;
```

1.6. Opérateurs logiques

Un opérateur logique combine le résultat de deux conditions pour produire un résultat unique, ou inverse le résultat d'une condition unique.

Tous les exemples vus jusqu'ici ne spécifiaient qu'une condition dans la clause WHERE.

1.6.1. L'opérateur AND

Avec AND, les deux conditions doivent être VRAIES.

Exemple :

```
SQL> select nom, prenom, fonction, sal
        from emps
        where sal >= 3000;
        and fonction = 'Informaticien';
```

1.6.2. L'opérateur OR

Avec l'opérateur OR, l'une ou l'autre des deux conditions doit être VRAIE.

Exemple :

```
SQL> select nom, prenom, fonction, sal
        from emps
        where sal >= 3000
        or fonction = 'Informaticiem';
```

1.6.3. L'opérateur NOT

Exemple d'utilisation

```
SQL>  select nom, prenom, fonction
        from emps
        where fonction not in ('Magistrat', 'Chfdpt') ;
```

L'opérateur NOT s'utilise également avec d'autres opérateurs SQL comme BETWEEN, LIKE et NULL.

Exemples:

```
Where sal not between 2000 and 3000;
Where prenom not like '_a%';
Where prime is not null;
```

1.7. Règles de priorité

Ordre de priorité	Opérateurs
1	Tous les opérateurs de comparaison
2	NOT
3	AND
4	OR

Les parenthèses permettent de modifier les règles de priorité

```
SQL> select prenom, fonction, sal
     from emps
     where fonction = 'Economiste'
     or fonction = 'Recteur'
     and sal >= 5000;
```

L'exemple renferme deux conditions :

1ère condition : la fonction est Recteur et le salaire >= 5000
2ème condition : la fonction est Economiste.

L'ordre SELECT se lit comme suit : sélectionner la ligne si l'employé est Recteur et s'il gagne plus de 5000 ou s'il est Economiste.

On peut utiliser les parenthèses pour forcer la priorité.

```
SQL> select prenom, fonction, sal
     from emps
     where (fonction = 'Economiste'
     or fonction = 'Recteur')
     and sal >= 5000;
```

Cet exemple contient deux conditions :

1ère condition : la fonction est Economiste ou Recteur
2ème condition : le salaire est supérieur à 5000.

1.8. La clause ORDER BY

Les lignes trouvées pour une requête sont ramenées dans un ordre quelconque. ORDER BY sert à trier les lignes. Si vous l'utilisez, vous devez la placer en dernier dans l'ordre SELECT. On peut spécifier une expression ou un alias sur lesquelles les lignes seront triées.

Exemple:

```
SQL> select prenom, fonction, datemb
     from emps
     order by datemb;
```

Pour un tri décroissant, utiliser le mot clé DESC

Exemple :

```
SQL> select prenom, fonction, datemb
     from emps
     order by datemb desc ;
```

L'ordre de tri par défaut est ascendant (ASC). Les valeurs NULL sont affichées en dernier dans l'ordre croissant et en premier dans le cas décroissant.

1.9. Tri sur l'alias d'une colonne

On peut utiliser un alias de colonne dans la clause ORDRER BY. Dans l'exemple ci-dessous, les données sont triées par salairannuel

```
SQL> select numemp, nom, prenom,
     sal*12 Salairannuel
     from emps
     order by Salairannuel;
```

1.10. Tri sur plusieurs colonnes

On peut trier les résultats d'une requête sur plusieurs colonnes, à concurrence du nombre de colonnes présentes dans la table concernée.

Dans la clause ORDER BY, spécifier les noms de colonnes en les séparant par une virgule. On peut effectuer un tri sur des colonnes non insérées dans l'ordre SELECT.

Examples:

```
SQL> select nom, prenom, numdept
     from emps
     order by numdept, sal desc;
```

```
SQL> select  prenom, sal
     from emps
     order by numdept, sal desc;
```

Exercices 1

1.1. Vrai ou faux les commandes SQL accèdent aux bases de données ?

1.2. L'ordre SQL suivant sera convenablement exécuté vrai ou faux?

```
Select nom, fonction, sal salaire
from emps;
```

1.3. Vrai ou faux l'ordre select suivant sera correctement exécuté ?

```
Select *
from depts ;
```

1.4. L'ordre SQL suivant comporte cinq erreurs

```
Select numemp, prenom
Sal x 12 salaire annuel

Lesquelles ?
```

1.5. Afficher la structure de la table *depts*

1.6. Afficher toutes les données de la table *depts*

1.7. Afficher la structure de la table *emps*. Créer une requête pour afficher le matricule (numemp), le prénom la fonction et la date d'embauche de tous les employés. Enregistrer l'ordre SQL dans fichier nommé *exo_17*

1.8. Exécuter la requête enregistrée dans le fichier *exo_17*

1.9. Créer la requête qui affiche les différentes fonctions (sans doublons) de la table *emps*

1.10. Créer la requête qui affiche le prénom et le salaire des employés qui gagnent plus de 3000.

1.11. Créer la requête qui affiche le prénom et le numéro de département de l'employé dont le matricule (numemp) est 4211

1.12. Modifier *exo_17* de sorte à afficher le prénom et le salaire de tous les employés dont le salaire n'est pas compris entre 3500 et 1600. Enregistrer la nouvelle requête sous le nom *exo_112.*

1.13. Afficher le nom, le prénom, la fonction et la date d'embauche des employés embauchés entre le 8 mars 95 et le 1er juillet 98. Trier le résultat par date d'embauche croissante.

1.14. Afficher prénom et le numéro de département des employés des départements 1 et 3. Trier le résultat par ordre croissant des prénoms.

1.15. Modifier *exo_17* pour afficher les prénoms et les salaires des employés gagnant plus de 3000 unités monétaires et travaillant dans les départements 1 et 3. Nommer les colonnes *Employés* et *Salaire mensuel*, respectivement. Enregistrer le nouvel ordre SQL sous le nom *exo_115*.

1.16. Afficher le prénom et la date d'embauche de tous les employés entrés en 1991.

1.17. Afficher le prénom et la fonction des employés dont la référence du supérieur hiérarchique (Refsup) est indisponible.

1.18. Afficher le nom, le prénom, le salaire et la prime de tous les employés qui perçoivent une prime. Trier le résultat par ordre décroissant des salaires et des primes.

1.19. Afficher le prénom de tous les employés dont la deuxième lettre est a.

1.20. Afficher les employés dont le nom comporte deux *l* et travaillant dans le département 1 ou dont le supérieur hiérarchique (Refsup) est 4536.

2. Les fonctions mono-ligne

2.1. Objectifs :

A la fin de ce chapitre, l'apprenant doit être capable

- De décrire les différents types de fonctions SQL
- D'utiliser les fonctions caractères, numériques et dates dans les ordres SELECT
- D'expliquer les fonctions de conversion

Les fonctions rendent l'instruction SQL plus puissantes en permettant de manipuler des valeurs de données.

Les fonctions sont utilisées pour

- Effectue des calculs sur des données
- Transformer des données
- Effectuer des calculs sur des groupes de lignes
- Formater des dates et des nombres pour l'affichage
- Convertir des types de données de colonnes

Les fonctions SQL acceptent des arguments et ramènent des valeurs. Les fonctions mono-ligne ramènent une seule valeur par ligne de requête. Un argument peut être une valeur constante utilisateur, une variable, un nom de colonne ou une expression.

Les fonctions mono-ligne accepte un ou plusieurs argument(s), agissent sur chacune des lignes de la requête, ramènent un résultat par ligne ; peuvent ramener un résultat différent de celui mentionné, s'utilisent dans les clauses SELECT, WHERE et ORDER BY, et enfin peuvent être imbriquées.

2.2. Fonctions Caractère

2.2.1. Fonction de conversion majuscule/minuscule

Les fonctions mono-ligne caractères acceptent des données caractères en entré et ramènent des données caractères ou numériques.

Fonction	Modification
LOWER(colonne/expression)	Convertit les caractères alpha numériques en minuscule
UPPER(colonne/expression)	Convertit les caractères alpha numériques en majuscule
INITCAP(colonne/expression)	Convertit l'initiale de chaque mot en majuscule

2.2.2. Fonctions de manipulation de caractères

Fonction	Modification
CONCAT(colonne1|expression1, Colonne|expression2)	Concatène la 1ère chaine de caractères à la seconde.
SUBSTR(colonne|e xpression m [,n]	Extrait une partie de la chaine de caractères en commençant au caractère situé à la position m et sur une longueur de n caractères. Si m est une valeur négative, le décompte s'effectue dans le sens inverse (à partir du dernier caractère de la chaine).
LENGTH(colonne|expression)	Ramène le nombre de caractères d'une chaine
INSTR(colonne | expression, m)	Ramène une valeur égale à la position du caractère m
LPAD(colonne | expression, n, 'string')	Complète une chaine de caractères sur la gauche avec la chaine string pour parvenir à une longueur de n caractères.

Exemples	Resultats
SQL> select lower('NOUHOU SOW') from dual ;	nouhou sow
SQL> select upper('Nouhou Sow') from dual;	NOUHOU SOW
SQL> select initcap('nouhou sow') from dual;	Nouhou Sow

```
SQL> select 'La fonction de '||initcap(prenom)||' est  '||
        lower(fonction) as "Détails des emplois"
        from emps ;

SQL> select numemp, prenom, numdept
        from emps
        where prenom = 'talibé';

Résultat : zéro ligne sélectionnée.

SQL> select numemp, prenom, numdept
        from emps
        where lower(prenom) = 'talibé';

Résultat : une ligne sélectionnée.
```

L'exemple ci-dessus affiche le numéro de l'employé, son prénom talibé et le numéro de son département.

La clause WHERE du 1er ordre SQL spécifie le prénom de l'employé sous forme 'talibé'. Comme les prénoms dans la table emps sont stockés avec l'initiale en majuscule, il est impossible de trouver l'équivalent dans le talibé, et par conséquent de sélectionner des lignes.

La clause WHERE de l'exemple suivant indique que le prénom doit être convertit, l'initiale en majuscule pour être comparé à 'Talibé'. Les deux prénoms étant maintenant l'initiale en majuscule, la ligne correspondante sera ramenée.

```
SQL> select numenp, prenom, numdept
     from emps
     where intcap(prenom)='Talibé';
```

2.2.3. Fonctions de manipulation de caractère1

Fonctions	Résultats
SQL >select Concat('Oum','ou') form dual;	Oumou
SQL > select SUBSTR('Oumou',1,3) form dual;	Oum
SQL > select LENGTH('Oumou') form dual;	5
SQL > select INSTR('Oumou','m') form dual;	3
SQL > select LPAD(3000, 8, '*') form dual;	**** 3000

CONCAT	Concatène des valeurs. Avec CONCAT, le nombre de paramètres est limité à 2.
SUBSTR	Extrait une chaine de longueur fixe
INSTR	Fournit la valeur numérique correspondant à la position d'un caractère
LENGTH	Fournit la valeur numérique correspondant au nombre de caractère d'une chaine.
LPAD	Ajoute des caractères de remplissage à gauche d'une valeur alphanumérique qui sera ainsi cadrée à droite.

[11] Interroger la table factice DUAL .
Exemple : select contact ('Mam','adou') from DUAL;

2.3. Fonctions numériques

2.3.1. Descriptions

Fonctions	Descriptions
ROUND	Arrondit la valeur à la position spécifiée
TRUNC	Tronque la valeur à la position spécifiée
MOD	Ramène le reste d'une division

Exemples :

Commande	Résultat
SQL > Select Round(12.345,2) from dual ;	12.35
SQL > Select trunc(12.345, 2) from dual ;	12.34
SQL > Select mod(5, 2) from dual ;	1

2.3.2. Description détaillée

ROUND(colonne|expression, n)	Arrondit la valeur de la colonne ou de l'expression à une précision de 10^{-n}. Si n est positif, la valeur sera arrondie à n décimales. Si n est omis, il n'y aura pas de décimale. Si n est négatif, l'arrondi portera sur la partie du nombre située à gauche de la virgule (dizaine, centaine, etc.).
TRUNC(colonne|expression, n)	Tronque la valeur de la colonne ou de l'expression à une précision de 10^{-n}. Si n est positif, le nombre sera tronqué à n décimales. Si n est omis, il n'y aura pas de décimale. Si n est négatif, c'est la partie à gauche de la virgule (dizaine, centaine etc.) qui sera tronquée.
MOD(m, n)	Ramène le reste de la division de m par n.

2.3.3. Autres exemples

```
SQL> select round(35.923, 2), round(35.923, 0), round(35.923, -1)
     from dual;

     SQL> select trunc(35.923, 2), trunc(35.923), trunc(35.923, -1)
     from dual;

SQL> select prenom, sal, prime, mod(sal, prime)
     from emps
     where fonction = 'Informaticien';
```

2.4. Utilisation des dates

Oracle stock les dates dans un format numérique interne représentant le siècle, l'année le mois, le jour, les heures, les minutes et les seconds.

Le format d'entrée et d'affichage par défaut est dd-mon-yy

SYSDATE est une fonction qui permet d'obtenir la date et l'heure courante. SYSDATE s'utilise de la même façon qu'un nom de colonne quelconque. Il est usuel d'interroger la table factice DUAL. La table DUAL appartient à l'utilisateur SYS, mais tous les utilisateurs peuvent y accéder. Elle contient une seule colonne, DUMMY, et une seule ligne contenant la valeur X. La table DUAL est utilisée lorsqu'on désire ramener une valeur une seule fois, par exemple la valeur d'une constante, d'une pseudo colonne ou d'une expression qui ne dépend pas d'une table utilisateur.

Exemple : afficher la date courante au moyen de la table DUAL

```
SQL> select sysdate
from dual;
```

2.4.1. Opérations arithmétiques sur les dates

Comme la base de données stocke les dates sous forme numérique, il est possible d'effectuer des calculs tels que l'addition ou la soustraction au moyen d'opérations arithmétiques. Il est possible d'ajouter ou de soustraire des constantes numériques et des dates.

Opération	Résultat	Description
Date + nombre de jours	Date	Ajoute un certain nombre de jours à une date
Date - nombre de jours	Date	Soustrait un certain nombre de jours à une date
Date – date	Nombre de jours	Soustrait une date d'une autre
Date + (nombre d'heures)/24	Date	Ajoute un certain nombre d'heures à une date

Exemple :

```
SQL> select nom, prenom, (sysdate-datemb)/7 semaines
  from emps
  where numdept = 1;
```

L'ordre SELECT ci-dessus affiche le nom, le prénom et le nombre de semaines d'ancienneté des employés du département 1. La date d'embauche est soustraite de la date courante, puis le résultat est divisé par 7 pour obtenir le nombre de semaines.

2.4.2. Fonctions Date

Les fonctions dates s'appliquent au type de données Date. Toutes les fonctions date ramènent une valeur de type Date exceptée MONTHS_BETWEEN qui ramène une valeur numérique.

Months_Between(date1, date2) donne le nombre de mois entre deux dates, date1 et date2. Le résultat peut être positif ou négatif. Si date1 est postérieure à date2, le résultat est

positif. Si non il est négatif. La partie décimale du résultat représente une portion de mois.

Add_months(date, n) ajoute un nombre n de mois à une date. *n* doit être un nombre entier et peut être négatif.

Next_day(date, 'char') fournit la date de la première occurrence du jour spécifié ('char') après la date saisie. Char peut être un numéro de jour ou une chaine de caractères.

Last_day(date) indique la date du dernier jour du mois auquel appartient la date indiquée.

Round(date[,'fmt']) ramène la date, arrondie à l'unité précisée par le modèle de format fmt. Lorsque fmt est omis, la date est arrondie au jour le plus proche

Trunc(date[,'fmt']) ramène la date tronquée à l'unité précisée par le modèle de format fmt. Lorsque fmt est omis, la date est tronquée au jour

La date ou l'année sont des modèles de format

Exemples :

Commandes	Résultat
SQL>select Months_between('02/08/10', '06/07/09') from dual;	12,8709677
SQL>select Add_months('01/01/11', 6) from dual;	01/07/11
SQL>select Next_day('16/06/11', 'Lundi') from dual;	20/06/11
SQL>select Last_day('16/06/11') from dual;	30/06/11

Afficher le titre, le nom, le prénom, la date d'embauche et le nombre de mois d'ancienneté de tous les employés ayant moins de 190 mois de service. Reprendre la syntaxe pour ajouter la date de révision du salaire 9 mois après la date d'embauche, le 1er lundi suivant la date d'embauche et le dernier jour du mois d'embauche

```
SQL> select titre, nom, prenom, datemb,
        months_between(sysdate, datemb) anciennete
        from emps
        where months_betwen(sysdate, datemb) < 190;

SQL> select titre, nom, prenom, datemb,
        months_between(sysdate, datemb) Anciennete,
        add_months(datemb, 9) Revision,
        next_day(datemb, 'Lundi') "Ce lundi est"
        from emps
        where months_between(sysdate, datemb) < 190;
```

Les fonctions ROUND et TRUNC peuvent être utilisées avec des valeurs du type numérique ou date. Utilisées avec les dates, ces fonctions arrondissent ou tronquent au modèle de format spécifié. On peut par conséquent arrondir les dates au 1er jour du mois ou de l'année les plus proches.

Exemple :

Afficher les dates d'embauche des employés ayant été recrutés en 2016. Afficher le matricule (numemp), le nom, le prénom, la date d'embauche et le mois de début d'activité en se servant des fonctions ROUND et TRUNC.

```
SQL> select numemp, nom, prenom, datemb,
        round(datemb, 'month'), trunc(datemb, 'month')
        from emps
        where datemb like '%16';
```

2.5. Fonctions de conversion

Il existe deux types de conversion : la conversion implicite et la conversion explicite. Bien que la conversion implicite de type de données soit possible, il est recommandé d'effectuer la conversion explicite afin d'assurer une meilleure efficacité des ordres SQL.

2.5.1. Conversion de type de données explicites

SQL offre trois fonctions pour convertir une valeur d'un certain type de données dans un autre.

To_char(number|date [, 'fmt']), convertit un nombre ou une date en une chaine de caractères de type varchar2 et de format fmt

To_number(char), convertit une chaine de caractères en nombre.

To_date(char [, 'fmt']), convertit une chaine de caractères représentant une date au format fmt en une date Oracle. Lorsque fmt est omis, le format est dd-mon-yy

Fonction To_char avec les dates

To_char(date, 'fmt'). Le modèle de format

- Doit être placé entre simples quotes et différencie majuscules et minuscules
- Peut inclure tout élément valide de format date
- Comporte un élément fm (file mode) qui supprime les espaces de remplissage ou les zéros de tête.
- Est séparé de la valeur date par une virgule.

Exemple :

```
SQL> select nom, prenom,
        to_char(datemb, 'fmDD Month YYYY') Datembauche
        from emps;

SQL> select nom, prenom,
        to_char(datemb, 'dd month yyyy') Datembauche
        from emps;
```

2.5.2. Fonction NVL

Pour transformer une valeur NULL en une valeur réelle, on utilise la fonction NVL. Elle s'utilise avec des types de données numérique, date et caractère. Les types de données doivent correspondre.

- NVL(prime, 0)
- NVL(datemb, '02/03/14')
- NVL(fonction, 'Pas de fonction')

Exemple :

```
SQL> select numemp, titre, nom, prenom, sal, prime,
        (sal*12) + nvl(prime, 0)
        from emps;
```

La rémunération annuelle de chaque employé est calculée en multipliant le salaire mensuel par 12 puis en ajoutant la prime.

Lorsqu'un argument est null dans une expression arithmétique, le résultat est null.

Exemple :

```
SQL> select nom, prenom, sal, prime,
       (sal*12) + prime
       from emps;
```

Le résultat du calcul ci-dessus est renseigné uniquement pour les employés qui perçoivent une prime.

2.5.3. La fonction DECODE

Elle permet de décoder les expressions de la même manière que l'ordre logique IF – THEN – ELSE utilisé dans de nombreux langages. Elle décode l'expression après l'avoir comparée à chacune des valeurs de recherche (search). Si l'expression est identique à search, le résultat est ramené.

Si la valeur par défaut est omise, on obtient une valeur null chaque fois que la colonne ou expression ne correspond à aucune valeur de search.

Exemple :

```
SQL>select nom, prenom, fonction, sal,
       DECODE(Fonction, 'Magistrat', sal*1.12,
                        'Econnmiste', sal*1.15,
                        'Sociologue', sal*1.18,
                        'Informaticien', sal*1.25,
                        sal) Salr_augmenT
                        from emps ;
```

2.5.4. Imbrication des fonctions

Le niveau d'imbrication des fonctions mono-lignes est illimité. L'évaluation se fait du niveau le plus interne vers le niveau externe.

Exemple :

Montrer le (ou les) employé(s) dont la référence du supérieur n'est pas précisé (null).

```
select numemp, titre, nom, prenom,
nvl(to_char(refsup), 'N"a pas de supérieur') information
from emps
where refsup is null;
```

1ère évaluation : la fonction interne convertit une valeur numérique en chaine de caractères. Résultat = to_char(refsup).

2ème évaluation : la fonction externe remplace la valeur Null par une chaine NVL(Résultat = 'N'a pas de supérieur).

Autre exemple :

Afficher la date du 1er lundi qui tombe 5 mois après la date d'embauche. Le résultat doit se présenté sous la forme : Lundi, 12 aout 2016. Trier le résultat par date d'embauche.

```
SQL> select to_char(next_day(add_months(datemb, 5),
        'Lundi'), 'fmDay, dd month yyyy') "1er Lundi après 5
        mois"
        from emps
        order by datemb;
```

Exercices 2

2.1. Ecrire une requête qui affiche la date courante puis l'enregistrer sous le nom *Date_courante*.

2.2. Afficher le matricule (numemp), le nom, le prénom et le salaire majoré de 25% sous la forme d'un entier pour chaque employé de la table emps. Nommer la colonne *Nouveau_sal.* Enregistrer la requête sous le nom *Sal_nouveau.* Exécuter la requête à partir du fichier *Sal_nouveau.*

2.3. Modifier la requête *Sal_nouveau* par ajout de la colonne *Révision* qui égale le nouveau salaire diminué de l'ancien.

2.4. Afficher le nom, le prénom et la date d'embauche de chaque employé ainsi que la date de révision du salaire qui est le 1er vendredi après 7 mois d'activité. Nommer la colonne *Révision.* Les dates doivent apparaitre sous la forme *Jour, mois année* (exemple Vendredi, 12 juillet 2016).

2.5. Afficher le nom de chaque employé et le nombre de mois travaillés depuis la date d'embauche. Nommer la colonne *Durée_travail.* Classer le résultat selon l'ancienneté. Arrondir le nombre de mois à l'unité près.

2.6. Afficher les employés dont le prénom commence par A, F, ou M ainsi que la longueur du prénom. Attribuer à chaque colonne un nom bien approprié.

2.7. Afficher le nom, le prénom, la date d'embauche ainsi que le jour de la semaine où l'employé a débuté. Nommer la colonne *Jour_s.* Trier le résultat dans l'ordre des jours de la semaine à compter du lundi.

2.8. Afficher le nom, le prénom et le montant de la prime de chaque employé. Pour les employés n'ayant pas de prime, afficher «Pas d prim». Nommer la colonne *Prime_emp*.

3. Afficher les données issues de plusieurs tables

3.1. Objectifs

Au terme de ce chapitre, vous devez être capables

- D'écrire des ordres SELECT pour accéder aux données de plusieurs tables en utilisant des équijointures et des non équijointures ;
- Visualiser les données ne répondant pas aux conditions de jointures en utilisant des jointures externes ;
- Relier une table à elle-même.

3.2. Qu'est-ce qu'une jointure ?

Une jointure sert à extraire des données de plusieurs tables

```
Select table1.colonne, table2.colànne
       from table1, table2
       where table1.colonne1 = table2.colonne2;
```

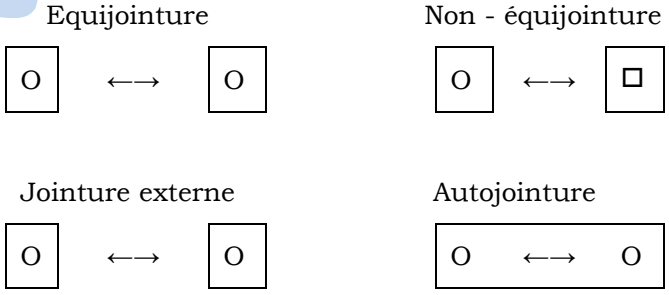
- table.colonne indique la table et la colonne d'où sont extraites les données ;
- table1.colonne1 = table2.colonne2 représente la condition qui joint les deux tables

Conseils :

- Par soucis de clarté et de facilité d'accès, il est recommandé de placer le nom de la table avant la colonne ;
- Lorsque le même nom de colonne apparait dans plusieurs tables, il doit obligatoirement être préfixé par le nom de la table

3.3. Types de jointures

Il existe deux types principaux de jointure : les équijointures et les non-équijointures.



Les autres méthodes de jointures sont les jointures externes, les autojointures et les opérateurs ensemblistes.

3.3.1. Equijointures

Pour déterminer le département auquel appartient un employé, il faut comparer les valeurs de la colonne Numdept de la table emps avec les valeurs de la colonne Numdept de la table depts. La relation établie entre les tables emps et depts est une équijointure : les valeurs de la colonne Numdept appartenant aux deux tables doivent être identiques. Ce type de relation fait souvent appel aux clés primaire et étrangère.

Nb : les équijointures sont aussi appelées jointures simples ou jointures internes.

- Extraction d'enregistrements avec les équijointures

```
SQL> select emps.numemp, emps.nom, emps.prenom,
        depts.zone
        from emps, depts
        where emps.numdept = depts.numdept;
```

Dans l'exemple ci-dessus, la clause SELECT spécifie les noms des colonnes à extraire. La clause FROM spécifie les deux tables de la base auxquelles on souhaite accéder. La clause WHERE spécifie la façon dont les deux tables sont jointes :

emps.numdept = depts.numdept

La colonne *numdept* étant commune aux deux tables, il faut la préfixer par le nom de la table d'appartenance afin de prévenir toute ambigüité.

- Ajout de condition de recherche

Pour afficher le matricule, le prénom, le numéro de département et la zone de l'employé *Balla,* on ajoute une condition dans la clause WHERE.

```
SQL> select emps.numemp, emps.nom, emps.prenom,
        depts.zone
        from emps, depts
        where emps.numdept = depts.numdept
        and prenom = 'Balla';
```

- Utilisation d'alias de table

La qualification des noms de colonne à l'aide des noms des tables prend beaucoup de temps, en particulier si les noms des tables sont longs. On peut substituer des alias aux noms des tables. De la même façon qu'un alias de colonne renomme une colonne, un alias de table donne un nouveau nom à une table. Les alias de table permettent ainsi de réduire le volume de code SQL et donc, de gagner de la place en mémoire.

```
SQL> select emps.numemp, emps.nom, emps.prenom,
        depts.numdept, emps.numdept, depts.zone
        from emps, depts
        where emps.numdept = depts.numdept;
```

```
SQL> select e.numemp, e.nom, e.prenom,
        d.numdept, e.numdept, d.zone
        from emps e, depts d
        where e.numdept = d.numdept;
```

Noter la manière dont les alias de table sont identifiés dans la clause FROM de l'exemple ci-dessus. Le nom de la table spécifié en entier est suivi d'un espace puis de l'alias de table. *e* est l'alias de la table emps, *d* celui de la table depts.

Conseils :

- Un alias de table peut compter jusqu'à 30 caractères, mais il est préférable qu'il soit le plus court possible.
- Lorsqu'un alias est substitué à un nom de table, cette substitution doit s'opérer dans la totalité de l'ordre SELECT.
- Choisir des alias « parlants »

NB. : Un alias de table ne s'applique qu'à l'ordre SELECT courant.

3.3.2. Non-équijointures

La relation entre les tables *emps* et *gradesal* est une non-équijointure car aucune colonne de la table *emps* ne correspond à une autre de la table *gradesal*. La relation entre les deux est la suivante : les valeurs de la colonne sal de la table emps sont comprises entre celles des colonnes *salmin* et *salmax* de la table *gradesal*.

L'exemple ci-dessous crée une non-équijointure pour évaluer l'échelon des salaires des employés. Le salaire est obligatoirement compris entre deux valeurs délimitant une tranche salariale.

```
SQL>select e.nom, e.prenom, e.sal, g.grade
        from emps e, gradesal g
        where e.sal between g.salmin and g.salmax;
```

Il est important de noter que tous les employés n'apparaissent qu'une seule fois dans la liste lorsque la requête est exécutée, et ceux pour deux raisons :

- Aucune ligne de la table des salaires ne déborde sur une autre. Autrement dit, le salaire d'un employé se situe entre la valeur minimale et la valeur maximale d'une ligne ;
- Tous les salaires des employés entrent dans les limites prévues par la table des échelons des salaires (*gradesal*). Aucun employé ne gagne moins que le salaire minimal de la colonne *Salmin* ni plus que le salaire maximal de colonne *Salmax*.

3.3.3. Jointures externes[2] : affichage d'enregistrements sans lien direct au moyen d'une jointure externe

Lorsqu'une ligne ne satisfait pas aux conditions de jointure, elle n'apparait pas dans le résultat de la requête. En exemple, lorsqu'on fait l'équijointure entre les tables *emps* et *depts,* le département *Langues* n'apparait pas car aucun employé n'est affecté à ce département.

```
SQL>select e.nom, e.prenom, e.numdept, d.nomdept
        from emps e, depts. d
        where e.numdept = d.numdept;[1]
```

Il est néanmoins possible de ramener la ou les lignes manquantes en plaçant un opérateur de jointure externe dans la condition de jointure. Cet opérateur se présente sous la forme d'un signe plus (+) inclus entre parenthèses, et placé du côté où l'information est incomplète.

Il crée une ou plusieurs lignes NULL, auxquelles une ou plusieurs lignes de la table complète peuvent être liée(s).

```
SQL>select e.nom, e.prenom, e.numdept, d.nomdept
        from emps e, depts. d
        where e.numdept(+) = d.numdept
        order by e.numdept;
```

[2] Insérer d'abord dans la table *depts* la ligne suivante : 5 Langues Sonfonia

Restrictions:

L'opérateur de jointure externe ne peut être place que d'un seul côté de l'expression, à savoir le coté où l'information manque;

Une condition comportant une jointure externe ne peut utiliser l'opérateur IN, ni être liée à une autre condition par l'opérateur OR.

3.3.4. Autojointures : liaison d'une table à elle-même.

Pour retrouver le nom et le prénom du supérieur de chaque employé, il faut que la table soit reliée à elle-même. Par exemple, pour retrouver le prénom du supérieur de l'employé *Balla,* il faut

Trouver *Balla* dans la table *emps* en recherchant dans la colonne PRENOM ;

Trouver le matricule du supérieur de *Balla* en cherchant dans la colonne REFSUP ;

Trouver le PRENOM du supérieur dont le matricule est 4834 dans la colonne NUMEMP puis regarder le PRENOM correspondant dans la colonne du même nom. Le matricule 4834 appartient à *Faza*, donc *Faza* est le supérieur de *Balla.*

```
SQL>select sh.nom, sh.prenom ||' est le superieur
          hierarchique de '||prof.nom, prof.prenom
          from emps prof, emps sh
          where prof.refsup = sh.numemp;
```

NB. : sh = supérieur hiérarchique

Exercices 3

3.1. Ecrire une requête qui affiche le n° de département, le nom et le prénom de chaque employé.

3.2. Créer la requête SQL qui affiche une liste unique de toutes les fonctions du département n° 3.

3.3. Afficher le nom, le prénom et la zone de tous les employés qui bénéficient d'une prime.

3.4. Afficher le nom, le prénom et le département des employés dont le prénom renferme la lettre *M.*

3.5. Afficher le nom, le prénom, la fonction et le n° de département des employés basés à Lambanyi.

3.6. Afficher le nom, le prénom et le matricule des supérieurs hiérarchiques et de leurs employés.

3.7. Modifier la requête en 3.4 pour afficher tous les employés y compris *Faza* qui n'a pas de « supérieur hiérarchique »

3.8. Afficher la structure de la table *emps*. Afficher le nom, le prénom, le département, le salaire et le grade de tous les employés.

3.9. Afficher le nom, le prénom et la date d'embauche de tous employés embauchés après *Balla.* Trier les lignes sur la date d'embauche.

3.10. Afficher le nom, le prénom, et la date d'embauche des employés ayant été recrutés avant leur supérieur. Nommer convenablement les colonnes.

4. Regrouper les données avec les fonctions de groupe

4.1. Objectifs :

A la fin de ce chapitre, vous devez savoir

- Identifier les fonctions de groupe
- Expliquer leur utilisation
- Regrouper les données avec la clause GROUP BY
- Inclure ou exclure des groupes de lignes avec la clause HAVING

4.2. Types de fonctions de groupe

Chaque fonction accepte un argument

Fonction	Description
AVG([distinct|all] n)	Valeur moyenne de n, en ignorant les valeurs NULL.
COUNT({*|[distinct|all] expr})	Nombre de lignes où expr est different de NULL. Le caractère * comptabilisent toutes les lignes sélectionnées y compris les doublons et les lignes NULL.
MAX([distinct|all] expr)	Valeur maximale de expr, en ignorant les valeurs null.
MIN([distinct|all] expr)	Valeur minimale de expr, en ignorant les valeurs null.
STDDEV([distinct|all] n)	Ecart standard de n, en ignorant les valeurs null.
SUM([distinct|all] n)	Somme des valeurs de n, en ignorant les valeurs null.
VARIANCE([distinct|all] n)	Variance de n, en ignorant les valeurs null.

NB. :

- Avec DISTINCT, la fonction prend en compte les valeurs distinctes. La valeur par défaut est ALL.
- Si expr est spécifiée, les différents types de données possibles pour les arguments sont Char, Varchar2, Number ou Date.
- Toutes les fonctions de groupe à l'exception de COUNT(*), ignorent les valeurs null.

4.2.1. Les fonctions AVG et SUM

AVG et SUM s'utilisent avec les données numériques.

Exemple :

```
SQL>select avg(sal), max(sal), min(sal)
        from emps
        where fonction = 'Informaticien';
```

4.2.2. Les fonctions MIN et MAX

MIN et MAX s'utilisent avec tout type de données.

Exemples :

- Afficher la date d'embauche de l'employé le plus récent et celle du plus ancien.

```
SQL>select min(datemb), max(datemb)
      from emps;
```

- Afficher le prénom des employés apparaissant en premier et en dernier dans le classement alphabétique

```
SQL>select min(prenom), max(prenom)
      from emps;
```

4.2.3. La fonction COUNT

- COUNT(*) ramène le nombre de lignes d'une table

```
SQL>select count(*)
      form emps ;
```

- COUNT(expr) ramène le nombre de lignes non null.

```
SQL>select count(prime)
      from emps
      where numdept = 3;
```

Autres exemples :

- Afficher le nombre de départements de la table emps.

```
SQL>select count(numdept)
      from emps;
```

- Afficher le nombre de départements distincts de la table emps.

```
SQL>select count(distinct(numdept))
      from emps;
```

4.3. Fonctions de groupe et valeurs null

Les fonctions de groupe ignorent les valeurs null des colonnes.

```
SQL>select avg(prime)
      from emps;
```

La fonction NVL force la prise en compte des valeurs null dans les fonctions de groupe.

```
SQL>select avg(nvl(prime, 0))
      from emps;
```

4.4. Création de groupes de données : la clause GROUP BY

La clause group by doit inclure toutes les colonnes de la liste SELECT qui ne figurent pas dans des fonctions de groupe.

Exemple :

```
SQL>select numdept, avg(sal)
      from emps
      group by numdept;
```

La colonne citée en group by peut ne pas figurer dans la liste SELECT.

Exemple :

```
SQL>select avg(sal)
        from emps
        group by numdept;
```

Dans la fonction de groupe on peut utiliser la clause ORDER BY.

Exemples :

```
SQL>select numdept, avg(sal)
        from emps
        group by numdept
        order by avg(sal);
```

```
SQL>select d.nomdept, round(avg(sal), 0) "Salr moy par dpt"
        from depts d, emps e
        where e.numdept = d.numdept
        group by d.nomdept;
```

Il est parfois nécessaire d'obtenir un résultat pour des sous-groupes de lignes. En exemple, montrer le total des salaires par fonction à l'intérieur de chaque département

```
SQL>select numdept, fonction, sum(sal) "Somme des salaires"
        from emps
        group by numdept, fonction
        order by numdept;
```

```
SQL>select d.nomdept, round(avg(sal), 0) SalMoyen
        from depts d, emps e
        where e.numdept = d.numdept
        group by d.nomdept
        order by d.nomdept;
```

4.5. Erreurs d'utilisation des fonctions de groupe dans une requête

4.5.1. Mélange des éléments individuels et des fonctions de groupe

```
SQL>select numdept, avg(sal)
        from emps;
```

Chaque fois qu'on mélange des éléments individuels (numdept) et des fonctions de groupe (avg), comme le montre l'ordre SELECT ci-dessus, on doit obligatoirement inclure une clause GROUP BY qui spécifie les éléments individuels (numdept) à grouper. Si la clause GROUP BY est omise, le message d'erreur « la fonction de groupe ne porte pas sur un groupe simple » apparait. La clause incorrecte est indiquée par un astérisque (*). L'erreur ci-dessus peut être corrigée en ajoutant la clause GROUP BY.

```
SQL>select numdept, avg(sal)
        from emps
        group by numdept;

SQL>select numdept, round(avg(sal), 2) salmoyen
        from emps
        group by numdept;
```

4.5.2. Utilisation de la clause HAVING

Il est impossible d'utiliser la clause WHERE pour limiter les groupes. On se sert plutôt de HAVING

```
SQL>select numdept, avg(sal)
        from emps
        where avg(sal) > 3000
        group by numdept;

        Erreur : fonction de groupe non autorisée ici
```

L'ordre SELECT correct est le suivant :

```
SQL>select numdept, avg(sal)
        from emps
        having avg(sal) > 3000
        group by numdept;

SQL>select fonction, sum(sal) "Somme salaire"
        from emps
        where fonction not like 'Info%'
        group by fonction
        having sum(sal) > 9500
        order by sum(sal);
```

4.6. Imbriquer des fonctions de groupe

Exemple:

Afficher le salaire moyen minimal

```
SQL>select min(avg(sal))
        from emps
        group by numdept;
```

Exercices 4

4.1. Vrai ou faux

- Les fonctions de groupe agissent sur plusieurs lignes pour produire un seul résultat ?
- Les fonctions de groupe intègrent les valeurs NULL dans les calculs ?
- La clause WHERE restreint les lignes avant qu'elles soient inclues dans les calculs de groupe ?

4.2. Afficher le salaire maximum, le salaire minimum, la somme des salaires et le salaire moyen. Nommer convenablement chaque colonne. Arrondir (quand c'est nécessaire) le résultat à l'unité près.

4.3. Modifier l'ordre SELECT ci-dessus pour afficher le salaire maximum, le salaire minimum, le total des salaires et salaire moyen pour chaque type de fonction. Enregistrer la requête sous le nom exo43.

4.4. Ecrire l'ordre SELECT qui affiche le nombre d'employés par fonction.

4.5. Déterminer le nombre d'employés ayant de subordonné(s) sans en donner la liste. Nommer la colonne « Nbre Chfs dpt ».

4.6. Ecrire l'ordre SELECT qui affiche la différence entre le salire maximum et le salaire minimum. Nommer la colonne « Ecarts »

4.7. Afficher le salaire des chefs de département et le plus bas salaire de leurs subordonnés. Exclure toutes lignes où le chef de département n'est pas identifié. Exclure tout groupe où le salaire minimum est inférieur à 3000. Trier le résultat par ordre décroissant sur les salaires.

4.8. Ecrire l'ordre SELECT qui affiche le nom du département, la zone, le nombre d'employés et le salaire moyen (arrondi à l'unité près) pour tous les employés de ce département. Nommer les colonnes Département, Nbre Employés et SalairMoyen respectivement.

4.9. Créer une requête pour afficher le nombre total d'employés puis, parmi ces employés, ceux qui ont été embauchés en 2010, 2015, 2016 et 2019. Nommer la colonne convenablement.

5. Les sous interrogations

5.1. Objectifs :

A la fin de ce chapitre, vous devez pouvoir

- Décrire les problèmes que les sous-interrogations peuvent résoudre.
- Définir des sous-interrogations.
- Enumérer les types de sous-interrogations.
- Ecrire des sous-interrogations mono et multi lignes.

5.2. Utilisation d'une sous-interrogation pour résoudre un problème

Admettons qu'on voudrait trouver quel employé gagne plus que *Balla*.

Pour résoudre un tel problème, deux requêtes sont nécessaire : une qui trouve le salaire de *Balla* et l'autre qui trouve le (ou les) employé(s) qui gagne(nt) plus que *Balla*.

A cet effet il faut combiner deux requêtes en plaçant l'une à l'intérieur de l'autre. La requête interne ou sou-interrogation ramène une valeur utilisée par la requête externe, ou principale. Utiliser une sous-interrogation, c'est exécuter deux requêtes successives en utilisant le résultat de la première comme valeur de recherche pour la seconde.

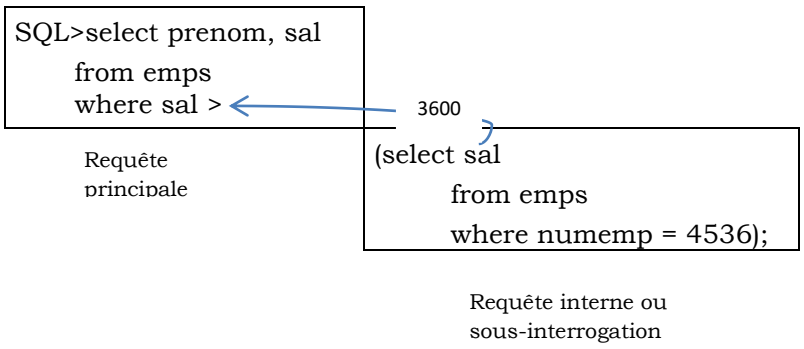
Les sous-interrogations s'avèrent très utiles pour sélectionner des lignes d'une table lorsqu'une condition dépend des données de la même table. On peut placer une sous-interrogation dans les clauses WHERE, HAVING et FROM.

Les opérateurs utilisés se classent en deux catégories :

- Les opérateurs mono lignes : >, =, >=, <, <= et <>.
- Les opérateurs multi lignes : IN, ANY et ALL.

Une sous-interrogation est souvent désignée par le nom « d'ordre select imbriqué », « sous ordre select » ou encore « ordre select interne »

Exemple:



Remarque sur l'utilisation des sous-interrogations

- Placer la sous-interrogation entre parenthèses.
- Placer la sous-interrogation à droite de l'opérateur de comparaison.
- Ne jamais ajouter de clause ORDER BY à une sous-interrogation.

- Utiliser les opérateurs mono-ligne avec les sous-interrogations mono-ligne.
- Utiliser les opérateurs multi-lignes avec les sous-interrogations multi-ligne.

5.3. Sous-interrogation mono-ligne

Une sous-interrogation mono-ligne ne ramène qu'une seule ligne. Elle utilise un opérateur mono-ligne

Exemple:

```
SQL>select nom, prenom, fonction
        from emps
        where fonction =
                        (select fonction
                        from emps
                        where numemp = 4439);
```

L'exemple ci-dessus affiche les employés occupant la même fonction que l'employé immatriculé 4439.

L'ordre SELECT ci-dessous affiche les employés occupant la même fonction que l'employé 4439 et gagnant un salaire (sal) supérieur à celui de l'employé 4321.

```
SQL>select nom, prenom, fonction
        from emps
        where fonction =
                        (select fonction
                        from emps
                        where numemp = 4439)
        and sal >
                        (select sal
                        from emps
                        where numemp = 4220);
```

Remarque:

Les requêtes internes et externes peuvent ramener des données issues de plusieurs tables.

5.3.1. Utilisation des fonctions de groupe dans une sous-interrogation

On peut afficher les données d'une requête principale en utilisant une fonction de groupe dans une sous-interrogation pour ramener une ligne unique. La sous-interrogation est inclue entre parenthèses et placée à la suite de l'opérateur de comparaison. L'exemple suivant affiche le nom, le prénom, la fonction et le salaire des employés dont le salaire est égal au salaire minimum. La fonction de groupe MIN transmet une valeur unique (1200) à la requête externe.

```
SQL>select nom, prenom, fonction, sal
        from emps
        where sal =  <-- 1200 --+
                                |
                        (select min(sal)
                        from emps);
```

5.3.2. La clause HAVING avec les sous-interrogations

En plus de clause WHERE, on peut utiliser des sous-interrogations dans la clause HAVING. Oracle exécute la sous-interrogation et ramène les résultats de clause HAVING de la requête principale.

En exemple,

- Afficher tous les départements dont le salaire minimum est supérieur au salaire minimum du département 2.

```
SQL>select numdept, min(sal)
        from emps
        group by numdept
        having min(sal) >
                        (select min(sal)
                        from emps
                        where numdept = 2);
```

  NB. : remplacer 2 (dans la sous-requête) par 1 puis observer de nouveau le résultat.

- Trouver la fonction ayant le salaire moyen le plus bas

```
SQL>select fonction, avg(sal)
        from emps
        group by fonction
        having avg(sal) =
                        (select min(avg(sal))
                        from emps
                        group by fonction);
```

5.3.3. Erreur liée aux sous-interrogations

L'affichage de plusieurs lignes dans une sous-requête en réponse à une sou-interrogation est une erreur courante.

```
SQL>select numemp, nom, prenom
        from emps
        where sal =
                        (select min(sal)
                        from emps
                        group by numdept);
```

Qu'est-ce qui ne va pas dans cette requête ?

Dans l'ordre SELECT ci-dessus, la sous-interrogation contient une clause *group by numdept,* qui implique que la sous-requête ramène plusieurs lignes, à savoir une par sous-groupe trouvé. Ici, le résultat de l'ordre select interne donnerait 1200, 1300 et 1450. La requête principale utilise le résultat de la sou-requête (1200, 1300 et 1450) dans sa clause WHERE. Celle-ci contient un opérateur égal (=), c'est-à-dire un opérateur de comparaison mono-ligne qui ne porte que sur une seule valeur. L'opérateur = ne pouvant accepter les valeurs multiples issues de la sou-interrogation, il en résulte une erreur dont voici le message :

sous-interrogation ramenant un enregistrement de plus d'une ligne

L'ordre SELECT suivant, va-t-il fonctionner ?

```
SQL>select numemp, nom, prenom, fonction
        from emps
        where fonction =
                        (select fonction
                        from emps
                        where prenom = 'Bala');
```

Qu'est-ce qui ne va pas ici ?

La sous-interrogation renferme la clause *where prenom = 'Bala'*. L'ordre SELECT semble correct mais aucune ligne n'est ramenée. En fait, aucun employé ne porte le prénom *Bala*. La sous-interrogation ramène alors zéro ligne, la requête principale ne ramènera, elle aussi, aucune ligne.

5.4. Sous-interrogations multi-ligne

Elles ramènent plusieurs lignes et s'utilisent avec un opérateur multi-ligne. Un opérateur multi-ligne porte sur une ou plusieurs valeurs.

Exemple :

Afficher les employés qui gagnent l'équivalent du salaire minimum d'un département.

```
SQL>select numdept, nom, prenom, sal
        from emps
        where sal in
                        (select min(sal)
                        from emps
                        group by numdept)
        order by numdept;
```

Dans l'ordre SELECT ci-dessus, remplacer min, dans la sous-interrogation, par max, puis observer le nouveau résultat.

L'opérateur ANY compare une valeur à chaque valeur ramenée par une sous-interrogation. L'exemple suivant montre les employés dont le salaire est supérieur à celui de n'importe quel Economiste et qui ne sont pas eux-mêmes Economiste. Le salaire minimal d'un Economiste est 1450. L'ordre SQL affiche alors tous les employés qui ne sont pas Economistes mais qui gagnent plus de 1450.

<ANY signifie inférieur à au moins une des valeurs.

>ANY signifie supérieur à au moins une des valeurs.

=ANY signifie équivalant à IN.

```
SQL>select numdept, prenom, fonction, sal
        from emps
        where sal > any
                        (select sal
                        from emps
                        where fonction = 'Economiste')
        and fonction <> 'Economiste';
```

L'opérateur ALL compare une valeur à toutes les valeurs ramenées par une sous interrogation. L'exemple ci-dessous affiche les employés dont le salaire est supérieur au salaire moyen de tous les départements. Le salaire moyen le plus élevé étant 2959.44, la requête renvoie tous les employés gagnant plus de 2959.44.

>ALL signifie supérieur au « maximum ».

<ALL signifie inférieur au « minimum ».

```
SQL>select numemp, prenom, fonction, sal
        from emps
        where sal > all
                        (select avg(sal)
                        from emps
                        group by numdept);
```

Exercices 5

5.1. Créer une requête SQL qui affiche le nom, le prénom et la date d'embauche de tous les employés travaillant dans le même département que *Balla*, excepté *Balla*.

5.2. Créer l'ordre SQL qui montre le matricule, le nom et le prénom de tous les employés qui gagnent un salaire supérieur à la moyenne.

5.3. Modifier l'ordre SELECT ci-dessus pour afficher le, le prénom et le salaire de tous les employés gagnant plus que le salaire moyen.

5.4. Afficher le nom, le prénom et le n° de département des employés dont le département est Lambanyi.

5.5. Afficher le matricule, le nom et le prénom des employés dont le supérieur est *Faza*.

5.6. Créer une requête pour afficher le nom, le prénom et le salaire des employés qui perçoivent un salaire supérieur à tout employé dont la fonction est Economiste. Trier le résultat par ordre décroissant des salaires.

6. Les sous-interrogations synchronisées

Une sous-interrogation synchronisée permet de lire chaque ligne d'une table et comparer la valeur de chaque ligne aux données associées. Elle est utilisée chaque fois qu'une sous-interrogation doit retourner un résultat ou un ensemble de résultats différents pour chaque ligne candidate prise en compte par la requête principale. En d'autres termes, une sous-interrogation synchronisée permet de répondre à une question à plusieurs choix, dont la réponse dépend de la valeur de chaque ligne traitée par l'ordre maitre.

Exemple :

Rechercher tous les employés dont le salaire est supérieur au salaire moyen de leur département.

```
select numdept, numemp, nom, prenom, sal
    from emps externe
    where sal >
                (select avg(sal)
                from emps interne
                where externe.numdept =
                interne.numdept)
    order by numdept;
```

La sous-interrogation synchronisée calcule explicitement le salaire moyen de chaque département. Comme les requêtes externe et interne utilisent toutes deux la table emps dans la clause FROM, un alias est attribué à chaque SELECT pour plus de clarté. Grace aux alias, les ordres SELECT deviennent plus lisibles et la requête s'exécute correctement car elle distingue la colonne de la table interne de celle de la table externe, ce qui impossible sans les alias

Utilisation de l'ordre EXISTS.

Avec des ordres SELECT imbriqués, tous les opérateurs logiques sont valides. De plus, on peut recourir à l'opérateur EXISTS. Ce dernier est souvent utilisé dans les sous-interrogations synchronisées, car il permet de tester l'existence d'une valeur. Si la valeur existe, il retourne la valeur TRUE ; sinon il retourne la valeur FALSE. De la même façon, l'opérateur NOT EXISTS garantie qu'il n'existe aucune valeur.

Exemple :

Rechercher tous les employés ayant au moins une personne sous leur responsabilité.

```
SQL>select nom, prenom, fonction
    from emps extern
    where exists
                (select numemp
                from emps intern
                where intern.refsup =
                extern.numemp);
```

Exercices 6

6.1. Ecrire une requête SQL pour afficher les quatre meilleurs salaires de la table emps. Afficher le nom, le prénom et le salaire des employés.

6.2. Ecrire une requête pour trouver tous les employés qui ne sont pas responsables. Afficher leur nom et leur prénom.

6.3. Ecrire la requête pour trouver les employés dont le salaire est supérieur au salaire moyen de leur département.

6.4. Ecrire une requête pour afficher les employés dont le salaire est inférieur

- A la moitié du salaire moyen de leur département.
- Au 2/3 du salaire moyen de leur département.

7. Mise en forme des résultats avec SQL*Plus

7.1. Objectifs :

A la fin de ce chapitre vous devez savoir

- Créer des requêtes avec des variables
- Personnaliser l'environnement SQL*Plus
- Afficher des résultats formatés

7.2. Etats interactifs

SQL*Plus permet de créer des états qui demandent à l'utilisateur de fournir lui-même des valeurs pour sélectionner les données ramenées. Pour de tels états, il faut placer des variables de substitution dans un fichier de commandes ou dans un ordre SQL.

7.2.1. Utilisation de la variable de substitution &

Lorsque vous utilisez, à l'intérieur d'un ordre SQL, une variable préfixée par l'esperluette (&), vous n'avez pas à définir la valeur de cette variable. L'ordre SQL suivant demande à l'utilisateur de saisir une fonction, puis affichera le nom, le matricule et la fonction des intéressés.

```
SQL>select numemp, nom, prenom, sal, fonction
      from emps
      where fonction = '&Fonction';
```

Dans la clause WHERE, les valeurs de type caractère et date doivent être saisies entre quotes (mais pas les valeurs numériques !)

Autre exemple:

```
SQL>select numemp, nom, prenom, sal
      from emps
      where numemp = &Numero;
```

7.2.2. Spécification de nom de colonne, d'expression et de texte lors de l'exécution

Les variables de substitution utilisables dans la clause WHERE peuvent également remplacer des noms de colonnes ou de tables, des expressions ou du texte.

Exemple :

Afficher le nom, le prénom et n'importe quelle autre colonne et condition relative aux employés.

NB. : lors de l'exécution de l'ordre ci-dessous, il faudra saisir une expression complète pour la condition sans oublier de mettre les données *caractère* et *date* entre simples quotes, exemples,

- *sal = 3000*
- *sal > 2999*
- *fonction = 'Chfdpt'* etc.

```
SQL>select nom, prenom, &nom_colonne
      from emps
      where &condition;
```

Autre exemple :

```
SQL>select numemp, nom, prenom, &nom_colonne
        from emps
        where &condition
order by &colonne_tri;

SQL>select numemp, nom, prenom, &nom_colonne
        from emps
        where &condition
        order by &colonne_tri desc;
```

7.3. Personnaliser l'environnement SQL*Plus

Pour contrôler la session courante, on utilise la commande SET

Syntaxe : SET SYSTEM_VARIABLE *value*

- system_variable = variable qui contrôle un aspect de l'environnement
- *value* = valeur de la variable système

7.3.1. Variables de la commande SET

Variables	Description
ARRAY[SISE]	Détermine le nombre de lignes qui seront ramenées de la base en une fois.
COLSEP{_|text}	Définit le texte à imprimer entre deux colonnes. Par défaut, le séparateur est un espace unique.
FEED[BACK] {6|off|on}	Affiche le nombre d'enregistrements ramenés par une requête lorsque celle-ci a au moins n.
HEA[DING] {off|on}	Détermine si les en-têtes de colonne apparaissent dans les états.
LIN[ESIZE] {80|n}	Fixe à n le nombre de caractères par lignes dans les états.
LONG {80|n}	Définit la longueur maximale d'affichage des valeurs du type LONG.
PAGES[ISE] {24|n}	Spécifie le nombre de lignes par page.
PAU[SE] {off|on}	Permet de contrôler le défilement de l'affichage sur le terminal. L'utilisateur doit appuyer sur Return pour voir la suite de l'affichage après chaque pause.

Remarque : n est une valeur numérique. Les valeurs soulignées sont les valeurs par défaut.

Pour voir la valeur de la variable de la commande SET, utiliser la commande SHOW, suivie du nom de la variable.

Exemples :

```
SQL>show feed
SQL>show lin
```

La commande SHOW ALL visualise toutes les variables à la fois.

7.3.2. Enregistrement d'une configuration personnalisée dans le fichier glogin.sql

Le fichier *glogin.sql* contient des commandes SET et d'autres commandes SQL*Plus qui s'exécutent lors de la connexion. Les valeurs implémentées par glogin peuvent être modifiées pendant la session

courante, mais les changements ne s'appliquent que durant cette session. Dès que vous quittez, les nouvelles valeurs sont perdues. Pérennisez les modifications en les plaçant dans le fichier *glogin.sql*.

7.3.3. Commandes de format SQL*Plus

7.3.3.1. La commande COLUMN

Syntaxe : Col[um] [{*Column*|*alias*} [Option]].

Cette commande contrôle le format des colonnes

- Cle[ar] efface le formatage de la colonne.
- For[mat] format modifie le format d'affichage d'une colonne.
- Hea[ding] text définit l'en-tête de colonne.
- Jus[tify] {align} cadre l'en-tête de colonne à gauche, au centre ou à droite.

Exemples :

- Définir les en-têtes de colonne :

  SQL> Column prenom hea 'Nom|Employé' for a20

  Le caractère (|) impose un saut de ligne si vous ne définissez pas de cadrage.

  SQL> Col sal jus left format $99,990.00

- Afficher le paramétrage de la colonne sal

  SQL>Col sal

- Effacer le paramétrage de la colonne sal :

  SQL>col sal clear

Pour afficher le paramétrage de toutes les colonnes, utiliser la commande COL.

Pour effacer le paramétrage de toutes les colonnes, lancer la commande CLE COL.

NB. : si la commande est longue, la poursuivre sur la ligne suivante en plaçant un tiret (-) à la fin de la ligne courante.

Modèles de format de la commande COLUMN

Modèle	Description	Exemple	Résultat
an	Définit une longueur de n.	a15	
9	Suppression des zéros de gauche	9999999	2345
0	Affiche les zéros de tête.	0999999	02345
$	Signe dollar flottant.	$9999	$2345
L	Symbole monétaire local flottant.	L9999	L2345
.	Position du point décimal	9999.99	2345.00
,	Séparateur de milliers.	9,999	2,345

Oracle affiche une chaine de dièses (#) à la place d'un nombre entier lorsque celui-ci contient plus de chiffres que ne l'autorise le modèle de format. De même, il affiche une chaine de dièses (#) à la place d'une valeur numérique dont le modèle est alphanumérique.

7.3.3.2. Commandes TTITLE et BTITLE

Syntaxes :

- Affichage d'en-tête de page : TTI[TLE] [*text*|*on*|*off*]

Exemple :

SQL>tti 'Etat des salaires des employés|Mai 2020'

- Définition d'un pied de page : BTI[TLE] [*text*|*on*|*off*]

Exemple :

SQL>bti 'Université UniC'

Remarques :

- Pour disposer l'en-tête ou le pied de page sur plusieurs lignes, utiliser le caractère (|).
- Les commandes TTITLE et BTITLE restent en vigueur jusqu'à leur désactivation ou tant que la session n'est pas fermée.
- Par défaut, TTITLE centre l'en-tête, affiche la date et le numéro de page en haut à gauche.

7.3.3.3. Utilisation de la commande NEW_VALUE dans la commande COLUMN

La commande NEW_V[ALUE] spécifie un nom de variable pour mémoriser le contenu d'une colonne.

```
SQL>COL numdept NEW_V ndpt for 9

SQL>tti SKIP 1 CENTER[3] 'Etat du département' ndpt-
      skip 2

SQL>BREAK ON numdept skip PAGE

SQL>select nom, prenom, refsup, numdept, sal
      from emps
      order by numdept;
```

L'exemple ci-dessus illustre la création d'un état dont les résultats sont répartis sur des pages séparées en fonction du département. L'option NEW_VALUE crée la variable (*ndpt*) associée au numéro de département, laquelle variable est ensuite utilisée dans l'ordre TTITLE en tant que partie intégrante de l'en-tête de chaque page de l'état. La commande BREAK ON crée une nouvelle section pour chaque département.

Résumé :

```
SQL> -- Le titre est placé à gauche (LEFT)
SQL> -- SKIP 1 saute une ligne
SQL> -- le caractère « - » en fin de ligne indique que TTITLE (ou
        la ligne) continue
SQL>
SQL>TTITLE LEFT '-------------------------' skip 1 -
> LEFT 'Employés et salaire' skip 1 -
> LEFT '-------------------------'
SQL>
SQL> -- L'exécution de l'ordre SQL provoque l'affichage du titre
SQL>
```

[3] Reprendre cet ordre SELECT en omettant l'option CENTER puis observer le résultat.

```
SQL>select prenom, sal
        from emps
        where numdept  = 1 ;
```

Résultat de l'exécution de la requête :

```
-------------------------
Employés et salaire
-------------------------
PRENOM        SAL
------------ ----------
Faza          5500
Balla         3650
Nouhou        3390
Nana          2500
Aissatou      2600
Aminata       3250
Talibe        3120
Naby          1300
Issa          1325

9 ligne(s) sélectionnée(s).
```

7.3.3.4. Utilisation des variables de la commande SET

```
SQL>SET NEW_PAGE 3

SQL>select numemp, nom, prenom, refsup
        from emps;
```

Dans l'exemple ci-dessus, SET NEW_PAGE insère 3 lignes blanches au début de chaque page du résultat.

7.3.4. La commande BREAK

- Supprimer les doublons et diviser les lignes en section :

  SQL>BREAK ON nom ON fonction

- Production des totaux généraux :

  SQL>BREAK ON report

- Saut de ligne(s) ou de page au niveau des ruptures :

  SQL>BREAK ON prenom SKIP 4 ON fonction SKIP 2

Utiliser la commande BREAK pour diviser les lignes en différentes sections et supprimer les doublons. Pour garantir que la commande BREAK fonctionne bien, il est nécessaire de trier le résultat de la requête sur la ou les colonnes de rupture (clause ORDER BY).

Syntaxe :

BREAK ON Column[*alias*|*row*] [SKIP *n*|*dup*|*page*] on … - [on report]

- *page* débute une nouvelle page lorsque la valeur de rupture change
- *skip n* saute n lignes lorsque la valeur de rupture change.
- *dup[icate]* affiche les doublons.

Un BREAK peut être défini sur les éléments suivants : colonne, ligne, page ou état.

Pour effacer tous les paramétrages définis par BREAK, utiliser la commande CLEAR BREAK.

7.3.5. La commande COMPUTE

La commande COMPUTE calcule et affiche les lignes des totaux

Syntaxe :

COMP[UTE] [Fonction [Label labelname] …
of {*expr*| *column*| *alias*} …
on {*expr*| *column*| *alias*| *Report*| *Form*}]

COMPUTE utilise les fonctions de calcul standard pour traiter et imprimer des totaux par sous-ensemble de lignes sélectionnées. Utilisée seule, COMPUTE permet d'afficher tous les calculs définis dans la session courante.

Fonction	Calcul	S'applique au type de données
AVG	Moyenne des valeurs non null.	Number
COUNT	Comptage des valeurs non null.	Tous les types
MAX	Valeur maximale	Number, char, varchar2, varchar
MIN	Valeur minimale	Number, char, varchar2, varchar
NUM	Comptage de lignes	Tous les types
STD	Ecart)type des valeurs non null	Number
SUM	Somme des valeurs non null	Number
VAR	Ecart des valeurs non null.	Number

Utilisation:

```
SQL>Break on fonction skip 2
SQL>Compute sum of sal on fonction
SQL>select fonction, titre, nom, prenom, sal
        from emps
        where fonction in ('Informaticien', 'Economiste',
        'Magistrat')
        order by fonction, sal;
```

Dans l'exemple ci-dessus, COMPUTE calcule et affiche le salaire total de chaque fonction citée dans la clause WHERE

On obtient la liste des employés de chaque fonction, avec le salaire total à la fin de chaque section. Il est possible de diviser le résultat en section dans toutes les colonnes de la table.

Dans la colonne fonction, la clause ORDER BY garantie le regroupement des postes pour chaque total.

NB. : la commande COMPUTE n'a aucun effet si la commande BREAK correspondante n'a pas été passée.

L'option SKIP de la commande BREAK permet d'insérer des lignes blanches entre le total et les lignes de détail du groupe suivant.

La ligne SUM générée par la commande COMPUTE représente le total des salaires pour les employés Informaticien, Economiste et Magistrat.

Utilisation de l'option LABEL avec COMPUTE

Dans l'exemple ci-dessous, l'option LABEL utilisée avec la commande COMPUTE permet de changer le libellé par défaut SUM généré par *Compute sum…*

```
SQL>Break on fonction skip 2
SQL>Compute sum of sal on fonction
SQL>select fonction, titre, nom, prenom, sal
        from emps
        where fonction in ('Informaticien', 'Economiste',
        'Magistrat')
        order by fonction, sal;
```

Exercises 7

7.1. Les tâches de la colonne de gauche ne correspondent pas aux commandes de la colonne de droite. Relier par une ligne les tâches correspondantes.

Tâche	Commande
Modifier les caractéristiques d'affichage d'une colonne	SET LINESIZE
Ajouter un titre à un état	BREAK
Calculer et imprimer des totaux	TITLE
Déterminer le nombre de lignes blanches en haut de page	COMPUTE
Contrôler le nombre de caractères par ligne	COLUMN
Forcer un saut de page à une rupture	SET NEW_PAGE

7.2. Ecrire un script le nom, le prénom la fonction et le nom du département des employés travaillant dans un département donné. La recherche s'effectuera indifféremment en minuscules ou en majuscules.

8. Extraction hiérarchique

8.1. Objectifs :

Au terme de ce chapitre, vous devez savoir

- Décrire le concept d'une requête hiérarchique.
- Créer un état sous forme d'arbre.
- Mettre en forme des données hiérarchiques
- Exclure des branches de la structure arborescente.

Les requêtes hiérarchiques permettent d'extraire des données basées sur une relation hiérarchique entre les lignes d'une table.

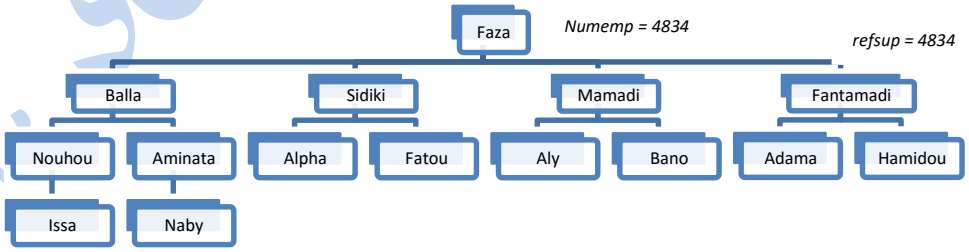
Une base de données relationnelle ne stocke les enregistrements de façon hiérarchique. Toutes fois, lorsqu'il existe une relation hiérarchique entre les lignes d'une table, le processus appelé parcours d'arbre permet d'établir une hiérarchie.

8.2. Structure arborescente

La table emps présente une structure arborescente qui représente la ligne des responsables. Pour créer une hiérarchie, utiliser le lien existant entre les valeurs équivalentes des colonnes numemp et refsup. Le numéro de responsable (refsup) d'un employé est lui-même le numéro d'employé de son supérieur.

La relation maitre-détail des structures arborescentes permet de déterminer

- La direction du parcours de la hiérarchie.
- Le point de départ de la hiérarchie.



8.3. Syntaxe hiérarchique

```
Select [Level], Column, expr…
        From table
        [Where Condition(s)]
        [Start whith Condition(s)]
        [Connect by prior Condition(s)];
```

Mots-clés et clauses

LEVEL	Pseudo colonne, LEVEL retourne le niveau 1 pour le nœud racine ; lorsqu'il vaut 2, il représente un enfant de la racine etc. LEVEL indique le numéro de niveau hiérarchique de l'élément dans l'arbre.
WHERE	Restreint les lignes ramenées par la requête sans affecter les autres lignes de la hiérarchie.
START WITH	Spécifie les lignes racines de la hiérarchie (point de départ). Cette clause est obligatoire pour une vraie requête hiérarchique.
CONNECT BY	Indique où il existe une relation entre des lignes parents et enfants.
PRIOR	Cette clause est obligatoire pour une requête hiérarchique. Prior indique la direction du parcours de l'arbre. Il permet également d'éliminer certaines branches de l'arborescence.

8.4. Parcours de l'arbre

8.4.1. Direction

– Du haut vers le bas	Column1 = clé parent Column2 = clé entant
– Du bas vers le haut	Column1 = clé enfant Column2 = clé parent

La direction peut aller du parent vers l'enfant ou inversement. Elle est indiquée dans la clause CONNECT BY par le positionnement du mot-clé PRIOR. La colonne citée derrière l'opérateur PRIOR indique quel est le parent. Exemple, dans la table *emps*, définir une relation hiérarchique dans laquelle la valeur de *numemp* de la ligne *parent* est égale à la valeur de *refsup* de la ligne *enfant*.

… connect by prior numemp = refsup

Parcours dans le sens inverse :

… connect by prior refsup = numemp

Il n'est pas nécessaire de coder l'opérateur PRIOR immédiatement après CONNECT BY. La formulation ci-après donne le même résultat que celui de l'exemple ci-dessus.

… connect by numemp = prior refsup

8.4.2. Point de départ

Il indique la condition à remplir et accepte tous les prédicats valides. En exemple, sur la table emps, commencer par l'employé *Fantamadi*.

… start with prenom = 'Fantamadi'

La clause START WITH détermine la ligne à utiliser à la racine de l'arbre. Cette clause peut être associée à n'importe quel prédicat valide. Exemple,

… start with refsup is null

Une expression peut porter sur plus d'une colonne. Dans l'exemple suivant, les valeurs de numemp sont évaluées pour la ligne *parent* et les valeurs de refsup, sal et prime sont évaluées pour les lignes *enfant.* L'opérateur PRIOR s'applique seulement à la valeur de numemp.

… connect by prior numemp = refsup and sal > prime

8.5. Classement des lignes avec la pseudo colonne LEVEL

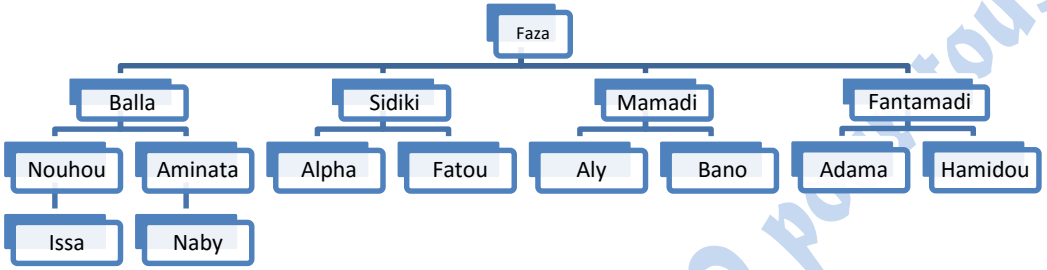
On peut explicitement afficher le niveau d'une ligne dans la hiérarchie à l'aide de la pseudo colonne LEVEL. L'état devient ainsi plus lisible. Les embranchements où la ligne principale se divise en une ou plusieurs branches sont appelés *nœuds*, et l'extrémité d'une branche est appelée *feuille* ou *nœud feuille*.

Le diagramme de la page suivante affiche les nœuds de l'arbre inversé ainsi que les valeurs de LEVEL. Dans ce schéma, Nouhou est à la fois *parent* et *enfant,* alors que l'employé Aly est un *enfant* et une *feuille.*

Pseudo colonne LEVEL

Valeur	Pour
1	un nœud
2	un enfant d'un nœud racine
3	un enfant d'un enfant
…	

Un *nœud racine* est le plus haut d'un arbre inversé. Tous les nœuds non racine sont des *nœuds enfants*. Tout nœud qui a des enfants est un *nœud parent*. Tout nœud sans enfant(s) est un *nœud feuille*.



Niveau 1 : racine/parent

Niveau 2 : parent/enfant

Niveau 3 : parent/enfant feuille

Niveau 4 : feuille

8.6. Formatage des états hiérarchiques avec LEVEL et LPAD

Créer un état affichant les niveaux hiérarchique de l'Université en commençant par le plus haut. Marquer chaque niveau par une indentation jusqu'au niveau le plus bas.

```
SQL>col Org_hierarchique for a16
SQL>select lpad('  ', 3 * level -3)||prenom Org_hierarchique,
        level, numemp, refsup, numdept
        from emps
        connect by prior numemp = refsup
        start with refsup is null;
```

Dans cet ordre SELECT, remplacer

- le premier *3* par le chiffre *4* puis exécuter.
- *a16* par *a20* puis de nouveau exécuter.
- *refsup is null* par *prenom = 'Faza'*
- *prenom = 'Faza'* par *numemp = 4834*

Quels constats faites-vous ?

Des numéros de niveau sont attribués aux nœuds d'un arbre à partir de la racine. La fonction LPAD associée à la pseudo colonne LEVEL permet d'afficher l'état hiérarchique sous la forma d'un arbre indenté.

LPAD(' ', 3 * level – 3) définit le format d'affichage des « prenom ». 3 * level – 3 est la longueur ayant pour chaine de caractères l'espace (le caractère blanc). Ceci indique à SQL de prendre le nombre de caractères composant le « prenom » puis le compléter à gauche par des caractères blancs jusqu'à atteindre une longueur égale à 3 * level – 3. Finalement chaque « prenom » sera décalé de 3 caractères blancs vers la droite par rapport celui du niveau supérieur.

8.7. Elagage des branches

Les clauses WHERE et CONNECT BY permettent d'élaguer l'arbre c'est-à-dire de déterminer les nœuds ou les lignes à afficher. Le prédicat opère comme une condition booléenne.

Exemple, en commençant à la racine, parcourir l'arbre de haut en bas et supprimer l'employé Aminata dans le résultat tout en traitant les lignes de détail.

```
SQL>col Org_hierarchique for a16
SQL>select lpad('  ', 3 * level -3)||prenom Org_hierarchique,
        level, numemp, refsup, numdept
        from emps
        where prenom != 'Aminata'
        connect by prior numemp = refsup
        start with refsup is null;
```

Autre exemple : en commençant à la racine, parcourir l'arbre de haut en bas et supprimer l'employé Aminata dans le résultat ainsi que les lignes de détail qui en dépendent.

```
SQL>select lpad('  ', 3 * level -3)||prenom Org_hierarchique,
        level, numemp, refsup, numdept
        from emps
        connect by prior numemp = refsup
        and prenom != 'Aminata'
        start with refsup is null;
```

8.8. Tri des données

Créer un état hiérarchique trié par numéro de département.

```
SQL>Break on numdept
SQL>select level, numdept, numemp, prenom, fonction
        from emps
        connect by prior numemp = refsup
        start with refsup is null
        order by numdept;
```

Il n'est pas conseillé d'utiliser la clause ORDER BY dans les requêtes hiérarchique, car on risque de détruire l'ordre normal implicite. N'utiliser cette clause qu'avec la pseudo colonne LEVEL.

Exercices 8

8.1. Créer un état représentant la hiérarchie des responsables par indentation. Afficher les prénoms et la référence du supérieur. Commencer par la plus haute personnalité.

8.2. Créer l'organigramme de la société représentant la hiérarchie des responsables. Commencer par la plus haute personnalité en excluant tout employé dont la fonction est Magistrat, ainsi que la branche de Sidiki.

9. Les transactions

Une transaction se compose des éléments suivants :

- Un ensemble d'ordres du LMD (Langage de Manipulation des Données) effectuant une modification cohérente des données
- Un ordre du LDD (Langage de Définition des Données)
- Un ordre du LCD (Langage de Contrôle des Données)

Les transactions offrent davantage de souplesse et un meilleur contrôle lors de la modification des données. Elles garantissent la cohérence des données en cas d'échec du processus utilisateur ou de panne du système.

Les transactions consistent en un ensemble d'ordres du LMD qui réalise une modification cohérente des données. Par exemple, un transfert de fonds entre deux comptes implique de débiter un compte et d'en créditer un autre du même montant. Les deux actions doivent soit réussir, soit échouer ensemble : un crédit ne peut être validé sans le débit correspondant.

Type de transaction	Description
LMD	Comprend un nombre quelconque d'ordres LMD qu'Oracle traite en tant qu'entité unique
LDD	Comprend un seul ordre LDD
LCD	Comprend un seul ordre LCD

Une transaction commence dès le premier ordre SQL exécuté et se termine lorsqu'un des évènements suivants se produit :

- Un ordre COMMIT ou un ROLLBACK est lancé.
- Un ordre LDD est lancé.
- Un ordre LCD est lancé.
- L'utilisateur quitte SQL.
- Il se produit une panne machine ou système.

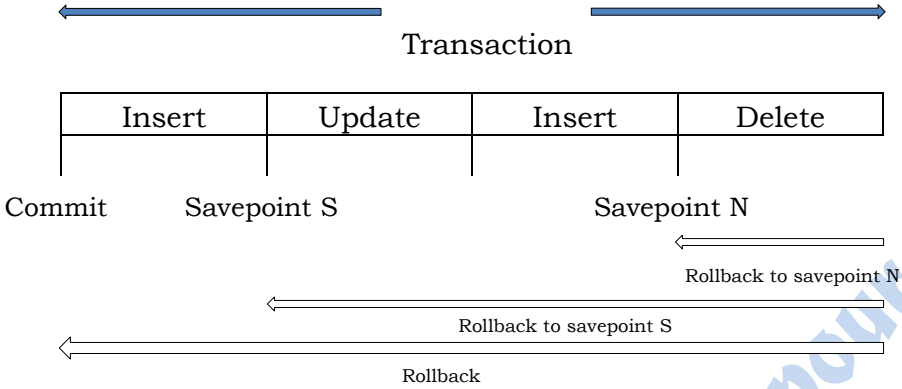
Lorsqu'une transaction prend fin, le prochain ordre exécutable démarre automatiquement la transaction suivante.

9.1. Avantage des ordres COMMIT et ROLLBACK

Les ordres COMMIT et ROLLBACK

- garantissent la cohérence des données.
- offre la possibilité d'afficher le résultat des modifications avant leur validation définitive.
- permettent un regroupement logique d'opérations.

9.2. Contrôle des transactions



On a la possibilité de contrôler la logique des transactions au moyen des ordres SAVEPOINT, COMMIT et ROLLBACK.

- COMMIT met fin à la transaction courante et rend définitive toute modification de données en instance.
- SAVEPOINT *name* pose une étiquette dans la transaction courante.
- ROLLBACK [TO SAVEPOINT *name*] met fin à la transaction courante et rejette toutes les modifications de données en instance. ROLLBACK TO SAVEPOINT *name* annule toutes les modifications jusqu'au SAVEPOINT et supprime celui-ci.

Remarque :

SQL*Plus propose la commande AUTOCOMMIT qui peut être activée (ON) ou désactivée (OFF). Lorsqu'elle est sur ON, chaque ordre du LMD est individuellement validé dès son exécution. Vous ne pouvez pas annuler les modifications.

Lorsqu'elle est sur OFF, vous pouvez préciser explicitement un COMMIT.

Panne système :

Lorsqu'une transaction est interrompue par une panne système, elle est annulée en totalité. Ainsi, SQL*Plus protège l'intégrité des données empêchant les modifications intempestives et en restaurant les tables dans l'état où elles se trouvaient lors de la dernière validation.

9.3. Etat des données avant COMMIT ou ROLLBACK

- Les opérations de manipulation de données se déroulant principalement dans le buffer de la base de données, il est possible de restaurer l'état précédent des données. Toutes les modifications effectuées au cours d'une transaction restent temporaires tant que la transaction n'est pas validée.
- L'utilisateur courant peut afficher le résultat de ces opérations de manipulation des données en interrogeant les tables.
- Les autres utilisateurs ne peuvent pas voir le résultat des opérations de manipulation de données réalisées par l'utilisateur courant. Oracle met en œuvre un principe de lecture cohérente qui garantit que l'utilisateur voit les données telles qu'elles se présentaient lors de la dernière validation.

- Les lignes concernées par la transaction sont verrouillées : *aucun utilisateur ne peut modifier les lignes qu'elle contient.*

9.4. Etat des données après COMMIT

Validation des modifications.

Pour enregistrer définitivement les modifications en instance, utiliser l'ordre COMMIT. Après exécution de ce dernier

- Les modifications de données sont écrites définitivement dans la base de données.
- L'état précédent des données est irrémédiablement perdu.
- Tous les utilisateurs peuvent voir le résultat de la transaction.
- Les lignes qui étaient verrouillées sont libérées et redeviennent accessibles à d'autres utilisateurs pour modification.
- Tous les SAVEPOINT sont effacés.

Validation des données

- Réaliser les modifications suivantes :

```
SQL>update emps
      set numdept = 20
      where numemp = 4011 ;
```

- Valider les modifications :

```
SQL>Commit ;
```

L'exemple ci-dessus met à jour la table emps en remplaçant par 20 le numéro de département de l'employé 4011. Il rend ensuite les modifications définitives en spécifiant l'ordre COMMIT.

9.5. Etat des données après ROLLBACK

Vous pouvez annuler toutes les modifications des données en instance au moyen de l'ordre ROLLBACK. Après un ROLLBACK,

- Les modifications apportées aux données sont annulées.
- Les données retrouvent leur état précédent.
- Les lignes verrouillées sont libérées.

Exemple :

En essayant de supprimer un enregistrement de la table emps, vous pouvez la vider accidentellement. Si cela vous arrive, corrigez l'erreur, spécifiez de nouveau l'ordre en veillant à ce qu'il soit correct, puis validez les modifications.

```
SQL>delete from emps;
SQL>rollback;
SQL>delete from emps
        where numemp = 4121;
SQL>select *
        from emps
        where numemp = 4121;
SQL>commit;
```

9.6. ROLLBACK au niveau ordre

Il est possible d'annuler une partie d'une transaction au moyen d'un ROLLBACK implicite si une erreur d'exécution d'un ordre est détectée. Si un seul ordre du LMD échoue dans la transaction, seul cet ordre est annulé. Oracle met en œuvre un SAVEPOINT implicite. Oracle émet un COMMIT implicite avant et après tout ordre du LDD. Toutes les autres modifications sont conservées.

L'utilisateur doit déterminer explicitement les transactions en exécutant un COMMIT ou un ROLLBACK

9.7. Lecture cohérente

On peut accéder à une base de données de deux manières différentes :

- En lecture (ordre SELECT)
- En écriture (ordres INERT, UPDATE, DELETE)

Le principe de lecture cohérente a les effets suivants :

- L'utilisateur qui lit la base de données et celui qui écrit ont une vue cohérente des données.
- Les utilisateurs qui accèdent en lecture ne peuvent pas voir les données en cours de modification.
- Les utilisateurs qui accèdent en écriture sont sûrs que leurs modifications seront cohérentes.
- Les modifications faites par un utilisateur ne peuvent pas interrompre ou gêner les modifications en cours d'un autre utilisateur.

L'objectif de la lecture cohérente est de garantir que chaque utilisateur voit les données telles qu'elles se présentaient lors de la dernière validation, c'est-à-dire avant le démarrage d'une opération du LMD.

La lecture cohérente s'applique automatiquement. Il consiste à conserver une copie partielle de la base de données dans des « *rollback segments* ». Lorsqu'une opération d'insertion, de mise à jour ou de suppression a lieu dans la base de données, Oracle copie les données avant leur modification dans un *rollback segment.*

Tous les utilisateurs, excepté celui à l'origine de la modification, continuent à voir les données telles qu'elles se présentaient avant les modifications. En réalité, ils voient un « cliché » des données copiées dans les rollback segments.

Tant que les modifications ne sont pas validées dans la base de données, seul l'auteur des mises à jour voit comment se présente la base de données une fois modifiée.

Après la validation d'un ordre du LMD, les modifications effectuées deviennent visibles par tout utilisateur en effectuant un SELECT.

L'espace occupé par les anciennes données dans les rollback segment est libéré et redevient disponible.

Si un ROLLBACK est exécuté, les modifications sont annulées et :

Les données d'origine présentent dans le rollback segment sont réécrites dans la table.

Tous les utilisateurs voient la base telle qu'elle était avant le début de la transaction.

9.8. Verrouillage

Les verrous sont des mécanismes qui empêchent les destructions dues aux interactions entre des transactions qui accèdent à la même ressource, qu'il s'agisse d'un objet utilisateur (comme des tables ou des lignes), ou d'objets systèmes non visibles par les utilisateurs (telles que des structures de données partagées et des lignes du dictionnaire de données).

Dans la base Oracle, le processus de verrouillage est entièrement automatique et ne requiert aucune intervention de l'utilisateur. Un verrouillage implicite a lieu pour tous les ordres SQL. Le mécanisme de verrouillage s'applique par défaut au niveau de restriction le plus bas possible, afin de permettre un degré élevé de concurrence transactionnelle tout en offrant une intégrité maximale des données.

10. Les contraintes

10.1. Objectifs :

A la fin de ce chapitre, vous devez savoir

- Définir les contraintes d'intégrité
- Créer des contraintes et les maintenir

Vous pouvez utiliser les contraintes pour

- Appliquer des règles au niveau d'une table chaque fois qu'une ligne est insérée, mise à jour ou supprimer dans cette table. La contrainte doit être satisfaite pour que l'opération réussisse.
- Empêcher la suppression d'une table s'il y a des dépendances avec d'autres tables.
- Fournir des règles pour des outils Oracle comme Developer 2000

10.2. Contraintes d'intégrité des données

Contrainte	Description
NOT NULL	Spécifie que cette colonne ne doit pas contenir de valeurs null.
UNIQUE	Spécifie une colonne ou une combinaison de colonnes dont les valeurs doivent être uniques pour toutes les lignes de la table.
PRIMARY KEY	Identifie chaque ligne de la table de manière unique.
FOREIGN KEY	Etablit et contrôle une relation de clé étrangère entre la colonne et une colonne de la table référencée.
CHECK	Spécifie une condition qui doit être vraie.

10.3. Conventions applicables aux contraintes

Toutes les contraintes sont stockées dans le dictionnaire de données. Elles seront très faciles à manipuler si vous leur donnez un nom parlant. Les noms des contraintes sont soumis aux conventions de dénomination des objets standards. Si vous ne nommez pas une contrainte, Oracle génère un nom dont le format est SYS_C*n* où *n* est un entier permettant de créer un nom de contrainte unique.

Il est possible de définir la contrainte au moment de la création de la table ou plus tard.

Syntaxe :

CREATE TABLE *[schema.] table*
(*Column datatype* [DEFAULT *expr*]
[*Column_constraint*],
…
[*table_constraint*]);

Dans cette syntaxe,

schema	Nom du propriétaire de la table
table	Nom de la table
DEFAULT *expr*	Valeur par défaut à utiliser si une valeur est omise dans l'ordre INSERT
Column	Nom de la colonne
datatype	Type de données et longueur de la colonne
Column_constraint	Contrainte d'intégrité incluse dans la définition de la colonne
table_constraint	Contrainte d'intégrité incluse dans la définition de la table

Exemple:

```
SQL>create table emps10(
        numemp number(4),
        nom varchar2(8),
        …
        numdept number(2) not null,
        constraint emps10_numemp_pk primary key
        (numemp));
```

On peut retrouver les contraintes définies pour une table spécifique en consultant la table USER_CONSTRAINTS du dictionnaire de données.

```
SQL>col owner a15
SQL>col table_name for a20
SQL>col table_name for a20
SQL>select owner, constraint_name, table_name,
        constraint_type
        from user_constraints
        where owner = 'SOWNH'
```

10.4. Les contraintes (niveaux)

En général, on crée les contraintes en même temps que la table. Il est cependant possible d'ajouter des contraintes ou d'en désactiver dans une table déjà créée. Il existe deux niveaux de contrainte :

10.4.1. Contrainte au niveau colonne

Elle référence une seule colonne et se définie lors de la spécification de cette colonne. Elle s'applique à n'importe quel type de contrainte d'intégrité.

column [CONSTRAINT constraint_name] constraint_type

10.4.2. Contrainte au niveau table

Elle référence une ou plusieurs colonnes et se définit indépendamment de la spécification de la (ou des) colonne(s). Elle est applicable à toutes les contraintes sauf NOT NULL.

Column, …

[CONSTRAINT constraint_name] constraint_type

(column, …),

10.5. La contrainte NOT NULL

```
SQL>create table emps11(
        numemp number(4),
        nom varchar2(8),
        prenom varchar2(15) not null,
        …
        numdept number(2) not null;
```

La contrainte NOT NULL ne peut être qu'au niveau colonne, pas au niveau table. L'exemple ci-dessus applique la contrainte NOT NULL aux colonnes prenom et numdept de la table emps11. Comme ces contraintes ne sont pas nommées, Oracle créera des noms pour elle.

On peut spécifier le nom de la contrainte pendant qu'on la défini.

```
        …
        numdept number(2) emps11_numdept_nn not null;
```

10.6. La contrainte de clé UNIQUE

Une contrainte d'intégrité de clé unique exige que chaque valeur dans une colonne ou dans un ensemble de colonne (la clé) soit unique, c'est-à-dire qu'elle n'existe pas dans plusieurs lignes pour la colonne ou l'ensemble des colonnes spécifiées. La colonne ou l'ensemble de colonnes indiqué(e) dans la définition de la contrainte UNIQUE constitue la *clé unique.* Si la colonne comprend plusieurs colonnes, le groupe de colonnes est appelé *clé unique composite.*

La contrainte UNIQUE autorise la saisie de valeurs null, à moins que vous ne définissiez également une contrainte not null sur les mêmes colonnes. En fait, lorsque des colonnes n'ont pas de contrainte NOT NULL, un nombre quelconque de lignes peuvent contenir de valeurs NULL puisque celles-ci n'équivalent à rien. Une valeur NULL dans une colonne (ou dans toutes les colonnes appartenant à une clé unique composée) satisfait toujours une contrainte de clé UNIQUE.

La contrainte de clé unique se définit au niveau colonne ou table.

```
SQL>create table depts1 (
        numdept number(1),
        nomdept varchar2(10),
        zone varchar2(15),
        constraint depts1_nomdept_uk unique(nomdept));
```

10.7. La contrainte PRIMARY KEY

Une contrainte clé primaire crée une clé primaire. Une seule clé primaire peut être créée par table. La contrainte PRIMARY KEY est une colonne ou un ensemble de colonnes qui identifie de manière unique chaque ligne d'une table. Elle établit une règle d'unicité de la colonne ou d'une combinaison de colonnes et garantit qu'aucune colonne faisant partie de la clé primaire ne contient de valeur NULL.

La contrainte clé primaire peut être définie au niveau table ou colonne. Une contrainte clé primaire composée se crée à l'aide de la définition de niveau table.

L'exemple ci-dessous crée une contrainte PRIMARY KEY pour la colonne NUMDEPT de la table *depts11*. Le nom de la contrainte est DEPTS11_NUMDEPT_PK.

```
SQL>create table depts11 (
        numdept number(1),
        nomdept varchar2(10),
        zone varchar2(15),
        constraint depts11_nomdept_uk unique(nomdept),
        constraint depts11_numdept_pk primary key(nomdept));
```

Un index UNIQUE est automatiquement créé pour une colonne PRIMARY KEY.

10.8. La contrainte FOREIGN KEY

La contrainte FOREIGN KEY, encore appelée contrainte d'intégrité référentielle, désigne une colonne ou une combinaison de colonnes comme étant une clé étrangère et établit une relation avec une clé primaire ou une clé unique de la même table ou d'une autre table. Une valeur de clé de clé étrangère doit obligatoirement correspondre à une valeur existante de la table maître ou être NULL. La contrainte FOREIGN KEY peut être définie au niveau table ou colonne. Une clé étrangère composée se crée au niveau table. L'exemple ci-dessous définit une contrainte FOREIGN KEY pour la colonne NUMDEPT de la table *emps11*. Le nom de la contrainte est EMPS11_NUMDEPT_FK.

```
SQL>create table emps11(
        numemp number(4),
        nom varchar2(8),
        …
        numdept number(2) not null,
        constraint emps11_numdept_fk foreign key (numdept)
        references depts11 (numdept));
```

La clé étrangère est définie dans la table détail (ou enfant) et la table contenant la colonne référencée est la table maître (ou parent). Pour définir la clé étrangère, on utilise les mots-clés suivants :

FOREIGN KEY	Définie une colonne de la table détail dans une contrainte de niveau table.
REFERENCES	Identifie la table et la colonne dans la table maître.
ON DELETE CASCADE	Indique que si une ligne est supprimée de la maître, les lignes dépendantes de la table détail seront également supprimées.

En l'absence de l'option ON DELETE CASCADE, la ligne de la table maître ne peut pas être supprimée si elle est référencée dans la table détail.

10.9. La contrainte CHECK

La contrainte CHECK définit une condition que chaque ligne doit obligatoirement satisfaire. La condition peut utiliser les mêmes constructions que les conditions d'une requête, aux exceptions près suivantes :

- Référence aux pseudo colonnes CURRVAL, NEXTVAL, LEVEL et ROWNUM.
- Appel aux fonctions SYSDATE, UID, USER et USERENV.

- Requêtes faisant référence à d'autres valeurs dans d'autres lignes.

Plusieurs contraintes CHECK peuvent être définies sur la même colonne. Le nombre de contraintes CHECK pouvant être associé à une colonne est illimité.

La contrainte CHECK peut être définie au niveau table ou colonne.

Exemple:

```
…
numdept number(2),
constraint emp11_numdept_ck
check (numdept between 1 and 9);
```

10.10. Ajout d'une contrainte

On peut ajouter une contrainte dans une table existante en utilisant l'ordre ALTER TABLE avec la clause ADD.

Syntaxe :

ALTER TABLE *table*
ADD [CONSTRAINT *constraint*] *type* (*column*);

table	nom de la table
constraint	nom de la contrainte
type	type de contrainte
column	nom de la colonne concernée par la contrainte

Il est recommandé de préciser le nom de la contrainte, même si ce n'est pas obligatoire. A défaut, le système génèrera lui-même des noms de contraintes.

Remarques :

- On peut ajouter, supprimer, activer ou désactiver une contrainte, mais il est impossible d'en modifier la structure.
- On peut ajouter une contrainte NOT NULL à une colonne existante en utilisant la clause MODIFY à la place de la clause ADD de l'ordre ALTER TABLE.
- On peut définir une colonne NOT NULL seulement si la table est vide ou bien si on spécifie une valeur par défaut, celle-ci sera alors automatiquement attribuée à toute les lignes existantes de la table.

Ajouter à la colonne *numemp* de la table *emps* une clé primaire nommée *emps_numemp_pk*.

```
SQL> alter table emps
        add constraint emps_numemp_pk primary
        key(numemp);
```

L'exemple ci-dessous crée une contrainte FOREIGN KEY dans la table *emps*. Cette contrainte vérifie qu'un supérieur hiérarchique existe en tant qu'employé valide dans la table *emps*.

```
SQL>alter table emps
        add constraint emps_refsup_fk
        foreign key(refsup) references emps(numemp);
```

10.11. Suppression d'une contrainte

Si vous souhaitez supprimer une contrainte, vous pouvez en retrouver le nom dans les vues USER_CONSTRAINTS et USER_COLUMNS du dictionnaire de données. Utilisez ensuite l'ordre ALTER TABLE avec la clause DROP. L'option CASCADE de la clause DROP provoque également la suppression de toutes les combinaisons associées.

Syntaxe :

ALTER TABLE *table*
DROP PRIMARY KEY|UNIQUE (*column*)|
CONSTRAINT *constraint* [CASCADE];

table	nom de la table
constraint	nom de la contrainte
column	nom de la colonne concernée par la contrainte

Exemples :

Supprimer la clé primaire de la table *depts11* ainsi que la clé étrangère associée définie sur la colonne *numdept* de la table *emps.*

```
SQL>alter table depts11
      drop primary key cascade;
```

Après vérification, annuler la suppression.

Supprimer la clé étrangère de colonne numdept de la table emps.

```
SQL>alter table emps11
      drop constraint emps11_numdept_fk;
```

Rétablir la clé après vérification.

10.12. Désactivation d'une contrainte

Pour désactiver une contrainte d'intégrité (ce qui évite d'avoir à la recréer), utiliser la clause DISABLE de l'ordre ALTER TABLE. Pour désactiver les contraintes d'intégrité dépendantes, ajouter l'option CASCADE

Syntaxe :

ALTER TABLE *table*
DISABLE CONSTRAINT *constraint* [CASCADE];

Exemple :

```
SQL>alter table emps
      disable constraint emps_numemp_pk cascade;
```

10.13. Activation d'une contrainte

Pour activer une contrainte d'intégrité actuellement désactivée dans la définition de la table, utiliser la clause ENABLE. En activant une contrainte UNIQUE ou PRIMARY KEY, un index correspondant est automatiquement créé.

Syntaxe :

```
ALTER TABLE table
ENABLE CONSTRAINT constraint ;
```

Exemple :

```
SQL>alter table emps
        enable constraint emps_numemp_pk;
```

NB. : si vous activer une contrainte, celle-ci s'applique à toutes les données de la table. Les données de la table doivent donc toutes respecter la contrainte.

10.14. Vérification des contraintes

Une fois que vous créez une table, vous pouvez en contrôler l'existence avec en exécutant la commande DESCRIBE. Cependant, la seule contrainte que vous pouvez vérifier de cette manière est NOT NULL. Si vous voulez afficher toutes les autres contraintes de votre table, interrogez alors la table USER_CONSTRAINTS.

L'exemple ci-dessous affiche toutes les contraintes de la table *emps*.

```
SQL>col search_condition for a20
SQL>select constraint_name, constraint_type, search_condition
        from user_constraints
        where table_name = 'EMPS';
```

NB. : en passant cet ordre SELECT, il faut respecter la casse de la table *emps*.

Remarque :

Les contraintes qui ne sont pas explicitement nommées par le propriétaire de la table se voient attribuer un nom par le système. Dans la colonne décrivant le type de contrainte, C désigne CHECK, P représente PRIMARY KEY, R intégrité référentielle et U, UNIQUE. A noter que la contrainte NOT NULL est en réalité une contrainte CHECK.

10.15. Affichage des colonnes associées aux contraintes

On peut afficher le nom des colonnes associées aux contraintes en interrogeant la table USER_CONS_COLUMNS du dictionnaire de données. Cette vue est particulièrement utile pour les contraintes dont le nom est attribué par le système.

```
SQL>col COLUMN_NAME for a15
SQL>select constraint_name, column_name
        from user_cons_columns
        where table_name = 'EMPS';
```

Exercices 10

10.1. Vérifier puis supprimer toutes les contraintes existantes.

10.2. Ajouter une contrainte PRIMARY KEY de niveau table à la table *emps* en utilisant la colonne *numemp*.

10.3. Ajouter une contrainte PRIMARY KEY à la table DEPTS en utilisant la colonne NUMDEPT.

10.4. Ajouter une clé étrangère à la table *emps* qui permet de contrôler qu'un employé n'est pas associé à un département inexistant.

10.5. Vérifier que les contraintes ont été ajoutées.

10.6. Vérifier le nom et le type d'objet correspondant aux tables *emps* et *depts*.

11. Création et gestion des tables

11.1. Objectifs :

A la fin de ce chapitre, vous saurez

- Décrire les principaux objets d'une base de données
- Créer des tables
- Décrire les différents types de données utilisables pour les définitions de colonne
- Modifier la définition des tables
- Supprimer, renommer et tronquer une table

11.2. Objets d'une base de données

Une base de données Oracle peut contenir de nombreuses structures de données. Chaque structure doit être prédéfinie lors de la conception de la base de données pour pouvoir être créée durant la phase de construction de la base.

- Table : stocke les données
- Vues : sous-groupe de données issues d'une ou de plusieurs tables
- Séquence : génère des valeurs de clé primaire
- Index : améliore la performance de certaines requêtes
- Synonyme : permet de donner un autre nom à un objet

11.3. Conventions de dénomination

Un nom

- Doit commencer par une lettre
- Peut comporter 1 à 30 caractères
- Ne peut contenir que les caractères A à Z, a à z, 0 à 9, _, $, et #
- Ne doit pas porter le nom d'un autre objet appartenant au même schéma
- Ne doit pas être un mot réservé Oracle

Conseils :

- Utiliser des noms parlants
- Utiliser une dénomination uniforme pour de entités identiques appartenant à des tables différentes. Par exemple, la colonne numéro de département s'appelle NUMDEPT dans la table *emps* et dans la table *depts*

Remarque :

Les majuscules et les minuscules ne sont pas différentes dans les noms. Exemple, EMPS est identique à Emps, emps, eMps etc.

11.4. L'ordre CREATE TABLE

Il fait partie d'une série d'ordres appartenant au LDD. Les ordres du LDD représentent un sous-ensemble d'ordres SQL utilisés pour créer,

modifier ou supprimer des structures de données oracle. Ils agissent directement sur la base de données, et enregistrent des informations dans le dictionnaire de données.

Syntaxe :

CREATE TABLE [*schema.*] *table*
(*column datatype* [DEFAULT *expr*], …

schema	Nom du propriétaire
table	Nom de la table
DEFAULT *expr*	Spécifie une valeur par défaut à utiliser en cas d'omission d'une valeur dans l'ordre INSERT.
column	Nom de la colonne
datatype	Type de donnée et longueur de la colonne.

Lorsqu'une table n'appartient pas à l'utilisateur, son nom doit être préfixé par le nom de son propriétaire (schéma).

On peut déclarer une valeur par défaut pour une colonne en utilisant l'option DEFAULT. Cette option empêche l'insertion de valeurs NULL dans une colonne lors de l'ajout d'une ligne qui ne comporte pas de données pour cette colonne. La valeur par défaut peut être un littéral, une expression ou une fonction SQL telle que SYSDATE, USER. Elle ne peut être le nom d'une autre colonne ni d'une pseudo colonne telle que NEXTVAL ou CURRVAL. L'expression par défaut doit correspondre au type de donnée de la colonne.

11.4.1. Création de tables

11.4.1.1. Créer la table

L'exemple ci-dessous crée la table DEPTS avec trois colonnes nommées NUMDEPT, NOMDEPT et ZONE.

```
SQL>create table depts(
      numdept number(1),
      nomdept varchar2(13),
      zone varchar2(15)
      );
```

L'exemple suivant crée la table EMPS avec dix.

```
SQL>create table emps(
      numemp number(4),
      titre varchar2(4),
      nom varchar2(10),
      prenom varchar2(15),
      fonction varchar2(15),
      refsup nmber(4),
      datemb date,
      sal number(6,2),
      prime number(6,2),
      numdept number(1)
      );
```

11.4.1.2. Vérification de la création avec la commande DES[CRIBE].

```
SQL>desc depts
SQL>desc emps
```

11.4.1.3. Interrogation du dictionnaire de données

Vous pouvez interroger les tables du dictionnaire de données pour afficher différents objets de la base vous appartenant. Les tables du dictionnaire les plus utilisées sont

- USER_TABLES
- USER_OBJECTS
- USER_CATALOG

Remarque : CAT est un synonyme de USER_CATALOG

Décrire les tables appartenant à un utilisateur

```
SQL>select *
      from user_tables ;
```

Afficher les différents types d'objets appartenant à l'utilisateur

```
SQL>select distinct object_type
      from user_objects;
```

Afficher les tables, les vues, les synonymes et les séquences appartenant à l'utilisateur

```
SQL>select *
      from cat;
```

11.4.1.4. Types de données

Type	Description
VARCHAR2(*size*)	Données caractères de longueur variable. Spécifiez obligatoirement une longueur (*size*) maximum. La longueur minimale est 1, la longueur maximale 4000
CHAR(*size*)	Données caractères de longueur fixe d'un nombre d'octets égal à *size*. La longueur minimum est par défaut 1 et la longueur maximum est 2000.
NUMBER(*p*,*s*)	Nombre de précision *p* et d'échelle *s*. La précision est le nombre total de chiffres allant de 1 à 38, et l'échelle est le nombre de chiffres à droite de la virgule allant de -84 à 127.
DATE	Valeurs de date et d'heures allant du 1er janvier 4712 avant J.C. au 31 décembre 9999 après J.C.
LONG	Données caractères de longueur variable jusqu'à 2 giga-octets
CLOB	Données caractères mono-octet jusqu'à 4 giga-octets.
RAW(*size*)	Données binaires de longueur *size*. La longueur maximum est de 2000. Spécifier obligatoirement une longueur maximale.
LONG RAW	Données binaires de longueur variable jusqu'à 2 giga-octets
BLOB	Données binaires jusqu'à 4 giga-octets.

BFILE	Données binaires stockées dans un fichier externe jusqu'à 4 giga-octets.

11.4.1.5. Création d'une table au moyen d'une sous-interrogation

Cette méthode permet à la fois de créer la table et d'y insérer des lignes ramenées par une sous-interrogation.

Syntaxe :

SQL>create TABLE *table*
[*column*(, *column*...)]
as *subquery*;

table	Nom de la table.
column	Nom de la colonne, valeur par défaut et contrainte d'intégrité.
Subquery	Ordre SELECT qui définit le groupe de lignes à insérer dans la nouvelle table.

L'exemple ci-dessous crée la table *Chefdpts* qui contient tous les employés dont la fonction est *Chfdpt* (chef de département). Remarquez que les données de la table *Chfdpts* proviennent de la table *emps.*

```
SQL>create table chfdpts
    as
      select titre, numemp, nom, prenom, sal*12 salranuel
      from emps
      where fonction = 'Chfdpt';
```

Vous pouvez vérifier l'existence de la table avec la commande desc.

Remarque :

- La table est créée avec les colonnes spécifiées puis remplies avec les lignes extraites au moyen de l'ordre SELECT.
- La définition de colonne ne peut contenir que le nom de la colonne et la valeur par défaut.
- Si des colonnes sont spécifiées, leur nombre doit être le même que celui de la sous-interrogation.
- Si aucune colonne n'est spécifiée, elles seront du même nom que celle de la sous-interrogation. Exemple,

```
SQL>create table informatique
    as
      select *
      from emps
      where fonction = 'Informaticien';
```

11.5. L'ordre ALTER TABLE

Après création d'une table, il peut arriver que vous ayez besoin de modifier la structure pour ajouter une colonne omise, changer une définition existante ou supprimer une colonne. Ceci est possible grâce à l'ordre ALTER TABLE.

11.5.1. Ajout d'une colonne

Syntaxe :

ALTER TABLE *table*
ADD (*column datatype* [DEFAULT *expr*]
[, *column datatype*]…);

table	Nom de la table
DEFAULT *expr*	Valeur par défaut de la nouvelle colonne
column	Nom de la nouvelle colonne
datatype	Type de donnée et longueur de la colonne.

Exemple :

Ajouter la colonne numdept à la table *chfdpts*.

```
SQL>alter table chfdpts
      add (numdept number(1));
```

Remarques :

- Vous ne pouvez pas choisir l'emplacement de la colonne: elle est systématiquement place à la fin de la table.
- Si une table renferme déjà des lignes lorsqu'on ajoute une colonne, la nouvelle colonne sera initialisée à NULL pour toutes les lignes sauf si l'on précise une valeur par défaut.

11.5.2. Modification de colonne

On peut modifier la définition d'une colonne au moyen de l'ordre ALTER TABLE et de la clause MODIFY. Les modifications peuvent être du type de donnée, de taille et valeur par défaut.

Syntaxe :

ALTER TABLE *table*
MODIFY (*column datatype* [DEFAULT *expr*]
[, *column datatype*]…);

Exemple:

Modifier la largeur de la colonne NOMDEPT de la table DEPTS pour la porter à 18.

```
SQL>alter table chfdpts
      mofify (nom varchar2(12));
```

Remarques :

- On peut augmenter la largeur ou la précision d'une colonne numérique.
- On peut réduire la largeur d'une colonne si celle-ci ne contient que des valeurs NULL ou si la table ne contient aucune ligne. Si elle contient des lignes, la largeur peut être réduite à concurrence du nombre de caractères de la donnée la plus longue.
- On peut modifier le type de donnée si la colonne contient des valeurs NULL ou si la table est vide

- Les modifications d'une valeur par défaut ne s'appliquent qu'aux insertions ultérieures.

11.5.3. Modification du nom d'une colonne

Au besoin, on peut modifier le nom d'une colonne par la commande ALTER TABLE.

Syntaxe:

ALTER TABLE *table*
RENAME column *old_column* to *new_column*

table	nom de la table
old_column	nom de la colonne à renommer
new_column	nom de la nouvelle colonne

11.5.4. Suppression de colonne

Pour supprimer une colonne d'une table, on utilise l'ordre ALTER TABLE avec la clause DROP.

Syntaxe:

ALTER TABLE *table*
DROP column *column*

table	nom de la table
column	nom de la colonne à supprimer

Exemple :

```
SQL> alter table chfdpts
       drop column titre;
```

11.6. Suppression de tables

L'ordre DROP TABLE supprime la définition d'une table Oracle. Lorsque vous supprimer une table, la base de données perd toutes les données de la table ainsi que tous les index associés.

Syntaxe :

DROP TABLE *table* ;
où

table	est le nom de la table

Exemple :

```
SQL>drop table chfdpts;[1]
```

Remarques :

- Toutes les données de la table sont supprimées.

- Les vues et les synonymes ne sont pas supprimés mais sont inutilisables.
- Toute transaction en instance est validée.
- Seul le propriétaire de la table ou un utilisateur ayant reçu le privilège DROP ANY TABLE peut supprimer une table.
- Une fois exécuté, l'ordre DROP TABLE est irréversible. Oracle ne demande pas confirmation lorsque vous lancez cet ordre

11.7. Modification du nom d'un objet

L'ordre RENAME est un ordre du LDD qui permet de renommer une table, une vue, une séquence ou un synonyme.

Syntaxe :

RENAME *old_name* TO *new_name*

old_name est l'ancien de l'objet

new_name est le nouveau de l'objet

Exemple :

Renommer la table *emps11* en *employes*

```
SQL>rename emps11 to employes ;
```

Remarque :

Vous devez être propriétaire de l'objet que vous renommer.

11.8. Vider une table

TRUNCATE TABLE est l'ordre du LDD qui permet de supprimer toutes les lignes d'une table en libérant l'espace de stockage. L'ordre est irréversible.

Syntaxe :

TRUNCATE TABLE *table*
où

table est le nom de la table

Exemple :

Vider la table *employes.*

```
SQL> truncate table employes;
```

Remarque :

Vous devez être propriétaire de la table ou disposer du privilège DELETE TABLE pour tronquer une table.

NB. : l'ordre DELETE supprime aussi les lignes d'une table, mais ne libère pas l'espace.

12. Manipulation des données

12.1. Objectifs

A la fin de ce chapitre, vous saurez

- Décrire chaque ordre du LMD
- Insérer des lignes dans une table
- Mettre à jour des lignes dans une table
- Supprimer des lignes d'une table
- Contrôler les transactions

12.2. Langage de manipulation de données

Le langage de manipulation des données joue un rôle central dans SQL. Chaque fois que vous *ajoutez*, *modifiez* ou *supprimer* des données dans la base, vous exécutez un *ordre du LMD. Un ensemble d'ordres du LMD groupés dans une unité de travail logique constitue une transaction.*

Considérons une base de données d'opérations bancaires. Quand un client de la banque transfert des fonds d'un compte d'épargne vers un compte courant, la transaction donne lieu à trois opérations différentes : débit du compte d'épargne, crédit du compte courant et enregistrement de la transaction dans le journal des transactions. Oracle doit garantir que les trois ordres sont exécutés pour maintenir la balance des comptes équilibrés. Quand quelque chose empêche la bonne exécution de l'un des ordres, les autres de la même transaction doivent être annulés.

12.3. L'ordre INSERT

Pour ajouter de nouvelles lignes dans une table, on utilise l'ordre INSERT.

Syntaxe :

INSERT INTO *table* [(*column* [, *column*...])]
VALUES (*value* [, *value*...]);

table	nom de la table
column	nom de la colonne dans la table à remplir
value	valeur qui figurera dans la colonne

12.3.1. Insertion de nouvelles lignes

Insérez une ligne dans la table *chfdpts* en précisant une valeur pour chaque colonne. Eventuellement, énumérez les colonnes dans la clause INSERT.

Exemples :

```
SQL>insert into chfdpts (numemp, nom, prenom, salranuel,
    numdept)
    values (5525, 'Kaba', 'Sekou', 2350, 1);
```

Remarquez que les valeurs sont insérées dans l'ordre des colonnes au sein de INSERT. Les valeurs du type caractère et date sont inclues entre simples quotes.

Lorsque des valeurs doivent être insérer pour toutes les colonnes, il n'est point indispensable de lister les colonnes. L'exemple suivant donne le même résultat que l'ordre INSERT ci-dessus.

```
SQL>insert into chfdpts
        values (5525, 'Kaba', 'Sekou', 2350, 1);
```

12.3.2. Insertion de lignes contenant des valeurs NULL

- Méthode implicite : on omet la colonne dans la liste

```
SQL>insert into chfdpts (numemp, nom, prenom, salranuel)
        values (5527, 'Soumah', 'Fatou', 2200);
```

- Méthode explicite : on spécifie le mot-clé NULL dans la liste VALUES tout en s'assurant que la colonne cible admet les valeurs NULL en vérifiant l'état NULL ? au moyen de la commande DESCRIBE.

```
SQL>insert into chfdpts
        values (5529, 'Balde', null, 2100, 2);
```

12.3.3. Insertion de valeurs spéciales

On peut utiliser les pseudo colonnes pour insérer des valeurs spéciales dans une table. L'exemple ci-dessous enregistre des informations sur l'employé *Boubacar* dans la table *Informatique*. Dans la colonne *datemb,* il insère la date et l'heure courante courantes issues de la fonction SYSDATE.

```
SQL>insert into informatique
        values(5530, 'M', 'Sow', 'Boubacar', 'Informaticien',
                4536, sysdate, 2800, null, 1);
```

12.3.4. Insertion de lignes au moyen de valeurs de substitution

Grâce aux valeurs de substitution, vous pouvez créer un ordre INSERT permettant à l'utilisateur d'ajouter des valeurs en mode interactif.

L'exemple ci-dessous enregistre des informations dans la table DEPTS. Il demande à l'utilisateur de saisir le numéro de département, le nom et la zone.

```
SQL>insert into depts (numdept, nomdept, zone)
        values (&ndpt, '&nomdept', '&zone');
```

NB. :

- Pour les valeurs du type date et caractère, l'esperluette (&) et la variable doivent être saisies entre simples quotes.
- Le nom de la variable peut être égal ou non à celui de la variable.

12.3.5. Copie des lignes d'une autre table

On peut utiliser l'ordre INSERT pour ajouter des lignes dans une table lorsque les valeurs proviennent de tables existantes. Dans ce cas, à la place de la clause VALUES, on se sert d'une sous-interrogation. Le

nombre de colonnes de la clause INSERT doit correspondre à celui de la sous-interrogation

Syntaxe :

INSERT INTO *table* [*column* (, *column*)]
subquery;

table	Représente le nom de la table à remplir
column	Représente le nom de la colonne dans la table à remplir
subquery	Représente la sous qui ramène les lignes de la table source.

Exemple :

Insérer dans la table *depts1* toutes les lignes de *depts* dont le nom de la zone commence par K.

```
SQL>insert into depts1(numdept, nomdept, zone)
      select numdept, nomdept, zone
      from depts
      where zone like 'K%' ;
```

12.4. L'ordre UPDATE

On peut modifier des lignes existantes au moyen de l'ordre UPDATE. Vérifier la mise à jour en interrogeant la table de manière à afficher les lignes modifiées.

Syntaxe :

UPDATE *table*
SET *column* = *value* [, *column* = *value*]
[WHERE *condition*];

table	est le nom de la table
column	est le nom de la colonne à modifier dans la table
value	est la nouvelle valeur qui figurera dans la colonne, ou une sous-interrogation fournissant cette valeur
condition	identifie les lignes à mettre à jour. Se compose de nom de colonne, d'expression, de colonne, de sous-interrogation et d'opérateurs de comparaison

12.4.1. Modification de lignes d'une table

Exemple :

Dans la table *emps1,* transférer l'employé *Barry Issa* dans le département 3 (Economie)

```
SQL>update emps1
      set numdept = 3
      where numemp = 4131;
```

La clause WHERE permet de modifier une ou plusieurs lignes spécifiques. Si vous omettez la clause WHERE, toutes les lignes seront modifiées.

Conseil :

Utilisez la clé primaire pour identifier une ligne unique. L'utilisation d'autres lignes comme critère de sélection risque de provoquer la modification de plusieurs lignes par inadvertance. En exemple, il est dangereux d'utiliser le nom d'employé pour désigner une seule ligne de la table *emps* car plusieurs employés peuvent avoir le même nom.

12.4.2. Modification avec une sous-interrogation multi-colonne

Exemple :

Modifier la fonction et le numéro de département de l'employé 4131 à l'identique de l'employé 4439.

```
SQL>update emps1
        set (fonction, numdept) =
                                (select fonction, numdept
                                from emps1
                                where numemp = 4439)
        where numemp = 4131;
```

12.4.3. Modification de lignes en fonction d'une autre table

Vous pouvez modifier les lignes d'une table en utilisant des sous interrogations dans les ordres UPDATE. Ainsi, l'exemple ci-dessous modifie la table *employes* en fonction des valeurs de la table *emps.*

L'exemple[4] ci-dessous modifie la table *employes* en fonction des valeurs de la table *emps.* Il mute tous les employés de la table *employes* ayant même fonction que l'employé *4439* dans le département de ce dernier.

```
SQL>update employes
        set ndept = (select numdept
                     from emps
                     where numemp = 4439)
        where fonction = (select fonction
                          from emps
                          where numemp = 4439);
```

Entraînez-vous pour mettre toutes autres lignes à jour.

12.4.4. Ordre UPDATE synchronisé

Syntaxe :

```
UPDATE table1  alias1
set column = (select expression
        from table2  alias2
        where alias1.column = alias2.column);
```

Dans le cas d'un ordre UPDATE, une sous interrogation synchronisée permet de mettre à jour les lignes d'une table basée sur des lignes d'une autre table.

Exemple :

Modifier la structure de la table *employes*, pour ajouter la colonne NOMDETP. Remplir ensuite les lignes de la table au moyen d'un ordre UPDATE synchronisé.

[4] Avant de passer à l'exemple, renommer la table emps1 *employes*, supprimer la colonne *numdept* puis ajouter la colonne *ndept* du type number, taille 1.

```
SQL>alter table employes
        add (nomdept varchar2(15)) ;

SQL>update employes e
        set nomdept = (select nomdept
                       from depts d
                       where e.numdept = d.numdept);
```

Dans cet ordre SELECT, bien veiller pour que la colonne à remplir ait le même nom que la colonne source.

12.4.5. Erreur de contrainte d'intégrité

```
SQL>update emps
        set numdept = 8
        where numdept = 1;
```

Si vous tentez de mettre à jour un enregistrement en utilisant une valeur ne respectant une contrainte d'intégrité, vous obtiendrez le message d'erreur suivant :

ORA-02291: violation de contrainte d'intégrité (SOWNH.EMPS_NUMDEPT_FK) - clé parent introuvable

Dans l'exemple ci-dessus, comme le numéro de département 8 n'existe pas dans la table maître DEPTS, il se produit une erreur de violation de clé maître.

12.5. L'ordre DELETE

Vous pouvez supprimer les lignes d'une table au moyen de l'ordre DELETE.

Syntaxe :

DELETE [FROM] *table*
[where *condition*];

table	nom de la table
condition	identifie la ou les lignes à supprimer. Se compose de noms de colonnes, d'expressions, de constantes, de sous-requête et ou d'opérateurs de comparaison

12.5.1. Suppression de lignes d'une table

Vous pouvez supprimer une ou plusieurs lignes spécifiques en précisant la clause WHERE dans l'ordre DELETE. Dans l'exemple qui suit[5], toutes les lignes de la table *dprtmt* sont supprimées. Vous pouvez le vérifier avec un ordre SELECT. La requête ne devrait retourner aucune ligne.

```
SQL>delete *
        from dprtmt;
```

Autres exemples :

- Supprimer dans la table *employes* tous les employés ayant été embauchés entre le 01/02/95 et le 01/01/98.

[5] Faire une copie de la table *depts* sous le nom *dprtmt* avant de lancer cette commande.

```
SQL>delete
        from employes
        where datemb between '01/02/95' and '01/01/98';
```

- Supprimer les employés dont la fonction est informaticien.

```
SQL>delete
        from employes
        where fonction = 'Informaticien';
```

12.5.1.1. Suppression de lignes en faisant référence à une autre table

L'exemple ci-dessous supprime tous les employés du département 3. La sous-interrogation recherche le numéro du département *Economie* dans la table *depts*. Elle transmet ensuite le numéro à la requête principale qui va supprimer les lignes de la table *employes* correspondant au numéro transmis.

```
SQL>delete from employes
        where numdept = (select numdept
                         from depts
                         where nomdept = 'Economie');
```

12.5.1.2. Ordre DELETE synchronisé

Avec un ordre DELETE, une sous-interrogation synchronisée permet de ne supprimer que des lignes existant dans une autre table

Syntaxe :

```
DELETE FROM table1  alias1
        WHERE column operator
                (SELECT expression
                FROM table2  alias2
                WHERE alias1.column = alias2.coloumn);
```

Exemple[6] :

Rechercher et supprimer tous les matricules doublons d'employées dans la table *employes.*

```
SQL>select prenom
        from employes externe
        where rowid > (select min(rowid)
                       from employes interne
                       where externe.numemp =
                       interne.numemp);
for update;

SQL>delete from employes externe
        where rowid > (select min(rowid)
                       from employes interne
                       where externe.numemp =
                       interne.numemp);
```

Remarque :

[6] Avant de lancer cet exemple, créer des lignes doublons dans la table *employes*

La clause FOR UPDATE verrouille les lignes ramenées par la requête. Les autres utilisateurs ne peuvent pas verrouiller ou mettre à jour les mêmes lignes tant que vous n'aurez pas terminé votre transaction.

Suppression de lignes : erreur de contrainte d'intégrité

Si vous tentez de mettre à jour un enregistrement en utilisant une valeur liée à une contrainte d'intégrité FOREIGN KEY, il se produit une erreur.

```
SQL>delete
        from depts
        where numdept = 2;
```

Dans l'exemple ci-dessus, on tente de supprimer le numéro de département 2 de la table DEPTS. Oracle affiche le message

ORA-02292: violation de contrainte (SOWNH.EMPS_NUMDEPT_FK) d'intégrité - enregistrement fils existant.

La tentative provoque une erreur car ce numéro est utilisé comme clé étrangère dans la table *emps*.

13. Création des vues

13.1. Objectifs :

A la fin de ce chapitre, vous saurez

- Décrire une vue
- Créer une vue
- Extraire des données par le biais d'une vue
- Modifier la définition d'une vue
- Insérer, mettre à jour et supprimer des données par une vue
- Supprimer une vue.

13.2. Qu'est-ce qu'une vue ?

On peut présenter des sous-ensembles ou des combinaisons de données en créant des vues sur les tables. Une vue est une table logique basée sur une table ou sur une autre vue. Elle affiche des données qui ne lui sont pas propres, c'est comme une fenêtre par laquelle il est possible de visualiser ou de modifier des données venant des tables. Les tables sur lesquelles une vue est basée sont appelées *tables de base.* Une vue est stockée dans le dictionnaire de données comme un ordre SELECT.

13.3. Avantages d'une vue

- Limite l'accès à la base de données en affichant une sélection de celle-ci.
- Permet aux utilisateurs de créer des requêtes simples pour extraire les résultats de requêtes complexes. Par exemple, vous pouvez extraire des informations de plusieurs tables sans savoir écrire un ordre de jointure.
- Garantit l'indépendance des données pour des utilisateurs et des programmes d'application contextuels. Une vue peut être utilisées pour extraire les données de plusieurs tables.
- Permet à des groupes d'utilisateurs d'accéder aux données en fonction de leurs critères particuliers.

13.4. Vues simples et vues complexes

On distingue les vues simples et les vues complexes. La différence principale entre ces deux types de vues est liée aux opérations LMD (insertion, mise à jour et suppression).

Une vue simple :

- Est dérivée des données d'une seule table
- Ne contient ni fonction ni groupes de données
- Permet d'exécuter des opérations du LMD

Une vue complexe :

- Est dérivée des données de plusieurs tables
- Contient des fonctions ou des groupes de données
- Ne permet pas toujours d'exécuter des opérations du LMD

13.5. Création d'une vue

Vous pouvez créer une vue en imbriquant une sous-interrogation dans l'ordre CREATE VIEW.

Syntaxe :

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
      [(alias [, alias]...)]
AS subquery
      [WITH CHECK OPTION [CONSTRAINT constraint]]
      [WITH READ ONLY]
```

OR REPLACE	recrée la vue, si celle-ci existe déjà.
FORCE	crée la vue, que les tables de base existent ou non.
NOFORCE	ne crée la vue que s'il existe des tables de base (valeur par défaut).
view	nom de la vue.
alias	indique les noms des expressions sélectionnées par la requête de la vue. Le nombre d'alias doit être égal au nombre d'expressions sélectionnées par la vue.
subquery	ordre SELECT complet. Vous pouvez utiliser des alias de colonnes dans la liste SELECT.
WTTH CHECK OPTION	n'autorise l'insertion et la mise à jour que pour les lignes auxquelles la vue peut accéder.
constraint	nom attribué à la contrainte CHECK OPTION.
WTTH READ ONLY	garantit qu'aucune opération LMD ne peut être exécutée dans la vue.

L'exemple ci-dessous illustre la création d'une vue contenant le numéro, le nom et la fonction de tous les employés du département 1.

```
SQL>create view empsvue1
      as select numemp, nom, fonction
      from emps
      where numdept = 1;
```

Vous pouvez afficher la structure de la vue à l'aide de la commande SQL*plus DESCRIBE.

13.6. Règles de création d'une vue

- La sous-interrogation qui définit la vue peut contenir une syntaxe SELECT complexe avec des jointures, des groupes et des sous-interrogations.
- Cette sous-interrogation ne peut pas contenir la clause ORDER BY que vous définissez lors de l'extraction des données de la vue.
- Si vous ne précisez pas un nom de contrainte pour une vue crée avec la contrainte CHECK OPTION, le système attribut un nom de contrainte par défaut au format SYS_c*n*.

- L'option OR REPLACE permet de modifier la définition d'une vue sans la supprimer ni la recréer, ou de lui accorder de nouveaux les privilèges objets qu'elle avait précédemment.

Vous pouvez contrôler les noms de colonnes en insérant des alias de colonnes dans sous-interrogation.

On peut aussi contrôler les noms de colonne en utilisant des alias de colonne dans la clause CREATE VIEW.

L'exemple ci-dessus illustre la création d'une vue avec l'alias NUM_EMPLOYE représentant le numéro des employés et l'alias SALARAIRE représentant le salaire, pour le département 3.

```
SQL>create view salvue3
        as select numemp num_employe, nom, sal salaire
        from emps
        where numdept = 3;
```

13.7. Extraction des données d'une vue

Les données d'une vue s'extraient de la même façon que les données d'une table. Vous pouvez afficher tout le contenu de la vue ou uniquement des lignes et des colonnes spécifique.

Exemple :

```
SQL>select *
        from salvue3;
```

13.8. Vues du dictionnaire de données

Une fois votre vue créée, vous pouvez interroger la table USER_VIEWS du dictionnaire de données pour afficher le nom et la définition de la vue. Le texte de l'ordre SELECT qui constitue votre vue est stocké dans une colonne de type LONG.

13.9. Accès aux données par une vue

Lorsque vous accédez à des données par le biais d'une vue, Oracle Server :

- Extrait la définition de la vue à partir de la table USER_VIEWS du dictionnaire de données.
- Vérifie les privilèges d'accès de la table de base de la vue.
- Convertit la requête de la vue en une opération équivalente dans la ou les tables de bases sous-jacentes. Autrement dit, les données sont extraites ou mises à jour dans la ou les tables de bases.

13.10. Modification d'une vue

L'option OR REPLACE permet de créer une vue. Même s'il en existe déjà une du même nom. Dans ce cas, la nouvelle vue remplace l'ancienne version de la vue pour son propriétaire. En d'autres termes, il est possible de modifier une vue sans avoir à la supprimer, la récréer ou lui accorder de nouveau les privilèges objet.

Exemple :

Modifier la vue *empsvue1* en ajoutant un alias pour chaque nom de colonne.

```
SQL>create or replace view empsvue1
        (numero_employe, nom_employe, titre_fonction)
        as select numemp, prenom, fonction
        from emps
        where numdept = 1;
```

Remarque :

Lorsque vous attribuez des alias de colonnes dans la clause CREATE VIEW, souvenez-vous que les alias doivent être cités dans le même ordre que les colonnes de la sous-interrogation.

13.11. Création d'une vue complexe

L'exemple ci-dessus illustre la création d'une vue complexe contenant les noms de départements, ainsi que les salaires minimum, maximum et moyens par département, à noter que des alias ont été définis pour cette vue, ce qui est obligatoire si une colonne de la vue est issue d'une fonction ou d'une expression.

```
SQL> create view depts_somm_vue(nomdpt, salmin, salmax,
        salmoy)
          as select d.nomdept, min(e.sal), max(e.sal), avg(e.sal)
        from emps e, depts d
        where e.numdept = d.numdept
        group by d.nomdept;
```

Vous pouvez afficher la structure de la vue à l'aide de la commande SQL* plus DESCRIBE. Pour afficher son contenu, utilisez un ordre SELECT.

13.12. Règles d'exécution des ordres du LMD dans une vue

Vous pouvez exécuter des opérations LMD sur les données d'une vue, sous réserve de respecter certaines règles.

Vous ne pouvez pas supprimer une ligne d'une vue si elle contient l'un des éléments suivants :

- Fonctions de groupe
- Clause GROUP BY
- Mot-clé DISTINCT

Vous ne pouvez pas modifier les données d'une vue qui contient une des conditions susmentionnées et un des éléments suivants :

- Colonnes définies par des expressions. Exemple : SALAIRE*12
- Colonne ROWNUM

Pour connaître la liste des colonnes modifiables d'une vue, vous pouvez consulter dans le dictionnaire de données : USER_UPDATABLE_COLUMNS.

Vous ne pouvez pas ajouter de données dans une vue si elle contient des éléments ci-dessus et s'il existe dans la table de base des colonnes NOT NULL sans valeur par défaut qui ne sont pas sélectionnées par la vue. Toutes les valeurs obligatoires doivent figurer dans la vue. N'oubliez pas que vous ajoutez des valeurs discrètement dans la table sous-jacente par les biais d'une vue.

13.13. Utilisation de la clause WITH CHECK OPTION

Il est possible d'exécuter des contrôles d'intégrité référentielle dans les vues. Vous pouvez ainsi renforcer les contraintes au niveau de la base de données. Une vue permet de protéger l'intégrité des données, mais dans un cadre très limité.

La clause WITH CHECK OPTION précise que les opérations d'insertion et de mise à jour exécutées dans une vue n'autorisent pas la création de ligne que la vue ne peut sélectionner. Par conséquent, elle permet de renforcer les contraintes d'intégrité et les contrôles de validation sur les données à insérer ou à mettre à jour.

Exemple :

```
SQL>create or replace view empsvue2
        as select *
        from emps
        where numdept = 2
        with check option constraint empsvue2_ck;
```

Si vous tentez d'exécuter des opérations LMD sur des lignes que la vue n'a pas sélectionnées, un message d'erreur s'affiche avec le nom de la contrainte, si ce dernier a été précisé.

```
SQL>update empsvue2
        set numdept = 5
        where numemp = 4011;
```

ERREUR à la ligne 1 :

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

Remarque :

Aucune ligne n'est mise à jour si le numéro du département est modifié, car la vue ne pourra plus afficher cet employé. Avec la clause WITH CHECK OPTION, la vue ne peut afficher que les employé du département 2 et n'autorise pas la modification du numéro de département de ces employés au travers de la vue.

13.14. Refus des ordres du LMD

L'option WITH READ ONLY vous permet de garantir qu'aucune opération du LMD ne sera exécutée dans votre vue. Dans l'exemple ci-dessous, la vue EMPSVUE1 est modifiée de telle sorte que sortes les opérations LMD sont interdites dans cette vue. Toute tentative de suppression d'une ligne dans la vue génère une erreur.

```
SQL>create or replace view empsvue1
        (numero_employe, nom_employe, titre_fonction)
        as select numemp, prenom, fonction
        from emps
        where numdept = 1
        with read only;
```

13.15. Suppression d'une vue

L'ordre DROP VIEW permet de supprimer une vue. La définition de la vue est ainsi supprimée de la base de données. La suppression d'une vue n'a aucun effet sur les tables sur lesquelles la vue est basée. En revanche, les vues ou autres applications basées sur des vues supprimées ne sont plus valides. Seul, le propriétaire ou un utilisateur ayant le privilège DROP ANY VIEW peut supprimer une vue.

Exercises 13

13.1. Créer la vue *emps_vue13* renfermant les matricules (numemp), les noms, les prénoms et les numéros de département. Modifier l'en-tête numdept en *numero_dprtmt.*

13.2. Afficher le contenu de la vue *emps_vue13*.

13.3. Afficher le nom de la vue (*emps_vue13*) et son « text » à partir de la table USER_VIEWS du dictionnaire de données.

13.4. Partant de la vue *emps_vue13*, afficher le prénom et le numéro de département des employés.

13.5. Créer la vue *depts_30* contenant le matricule (numdept), le nom, le prénom, la fonction et le numéro de département des employés du département 3. Renommer la colonne *numemp matricule.* La vue doit interdire toute mutation d'employé entre départements.

13.6. Afficher le contenu de la vue *depts_30*

13.7. Afficher la structure de la vue *depts_30*

13.8. Essayer de muter l'employé 4220 dans le département 1.

14. Autres objets de la base de données

14.1. Objectifs :

Au terme de ce chapitre, vous saurez

- Décrire d'autres objets de la base de données et leur utilisation
- Créer, mettre à jour et utiliser des séquences
- Créer et mettre à jour des index
- Créer des synonymes privés ou publics

14.2. La séquence – qu'est-ce que c'est ?

A l'aide d'un générateur de séquences, vous pouvez créer automatiquement des numéros de séquence pour les lignes des tables. Une séquence est un objet de données crée par un utilisateur et qui peut être partagé entre plusieurs utilisateurs.

En général, les séquences permettent de créer une valeur de clé primaire propre à chaque ligne. La séquence est générée puis incrémentée (ou décrémentée) par une routine interne d'Oracle. Vous pouvez ainsi gagner du temps, car le code applicatif utilisé pour écrire une routine de génération de séquences est considérablement réduit.

Les numéros de séquence sont stockés et générés indépendamment des tables. Vous pouvez donc utiliser la même séquence pour plusieurs tables.

14.2.1. Création d'une séquence

Générez automatiquement des numéros séquentiels à l'aide de l'ordre CREATE SEQUENCE.

Syntaxe :

```
CREATE SEQUENCE sequence
    INCREMENT BY n]
    START WITH n
    [{maxvalue|nomaxvalue}]
    [{minvalue|nominvalue}]
    [{CYCLE|NOCYCLE}]
    [{CAHE n|NOCACHE}];
```

sequence	Représente le nom du générateur de numéros séquentiels.
INCREMENT BY *n*	Définit l'intervalle entre les numéros de séquences, où *n* est un entier. Si cette clause est omise, le pas d'incrémentation est 1.
START WITH *n*	Indique le premier numéro de séquence à générer. Si cette clause est omise, la séquence commence par le numéro 1.
MAXVALUE *n*	Indique la valeur maximale que peut générer la séquence.
NOMAXVALUE	Indique la valeur maximale 10^{27} pour une séquence croissante, et la valeur -1 pour une séquence

	décroissante (option par défaut).
MINVALUE n	Indique la valeur minimale de la séquence.
NOMINVALUE	Définit la valeur minimale 1 pour une séquence croissante, et la valeur – (10^{26}) pour une séquence décroissante (option par défaut).
CYCLE|NOCYCLE	Précise si la séquence peut continuer à générer ou non des valeurs après avoir atteint sa valeur maximale ou minimale. NOCYCLE est l'option par défaut.
CACHE *n*|NOCACHE	Indique le nombre de valeurs qu'Oracle server préalloue et conserve en mémoire. Par défaut, Oracle server met 20 valeurs en mémoire cache.

L'exemple ci-dessous illustre la création de la séquence NUMDPT_DEPTS destinée à la colonne NUMDEPT de la table DEPTS. La séquence, qui commence avec le numéro 6, n'autorise ni la mise en mémoire cache ni le bouclage de la séquence. N'incluez pas l'option CYCLE si la séquence est destinée à générer des valeurs de la clé primaire, à moins que vous disposiez d'un mécanisme fiable qui purge les anciennes lignes plus rapidement que les cycles de la séquence.

```
SQL>create sequence numdpt_depts
        increment by 1
        start with 6
        maxvalue 50
        nocache
        nocycle;
```

14.2.2. Vérification des séquences.

Une fois votre séquence créée, elle est documentée dans le dictionnaire de données. Comme une séquence est un objet de la base de données, vous pouvez l'identifier dans la table USER_OBJECTS du dictionnaire de données.

Vous pouvez également vérifier les paramètres de la séquence en effectuant une sélection à partir de la table USER _SEQUENCES du dictionnaire de données.

```
SQL>select sequence_name, min_value, max_value,
        increment_by, last_number
        from user_sequences;
```

14.2.3. Utilisation d'une séquence

Une fois votre séquence créée, vous pouvez générer des numéros séquentiels pour vos tables. Référencez les valeurs de votre séquence à l'aide des pseudo colonnes NEXTVAL et CURRVAL

La pseudo colonne NEXTVAL permet d'extraire des numéros successifs à partir de la séquence indiquée. Vous devez désigner NEXTVAL avec le nom de la séquence. Lorsque vous référencez *sequence*, NEXTVAL, un nouveau numéro de séquence est généré et CURRVAL reçoit le numéro de séquence actuel. La pseudo colonne CURRVAL fait référence au numéro de séquence que l'utilisateur vient d'indiquer. NEXTVAL doit générer un numéro de séquence dans la session actuelle de l'utilisateur pour que CURRVAL puisse être référencée. Vous devez désigner CURRVAL avec le nom la séquence. Lorsque vous référencez *séquence,* CURRVAL, la dernière valeur retournée pour cette session utilisateur s'affiche.

14.2.4. Règles d'utilisation des pseudos colonnes NEXTVAL et CURRVAL

Vous pouvez inclure NEXTVAL et CURRVAL:

- En projection d'un ordre SELECT qui ne fait pas partie d'une sous-interrogative
- En projection d'une sous-interrogative dans un ordre INSERT
- Dans la clause VALUES d'un ordre INSERT
- Dans la clause SET d'un ordre UPDATE

Vous ne pouvez pas inclure NEXTVAL et CURRVAL dans :

- Un ordre SELECT de création d'une vue
- Un ordre SELECT comprenant le mot-clé DISTINCT
- Un ordre SELECT avec les clauses GROUP BY, HAVING et ORDER BY
- Une sous-interrogation dans un ordre SELECT, DELETE ou UPDATE
- Une expression DEFAULT dans un ordre CREATE TABLE ou ALTER TABLE

14.2.5. Utilisation d'une Séquence

L'exemple ci-dessous illustre l'insertion d'un nouveau département dans la table DEPTS. La séquence NUMDPT_DEPTS permet de générer un nouveau numéro de département.

```
SQL>insert into depts
        values(numdpt_depts.nextval, 'Finances', 'Kaloum');
```

Vous pouvez afficher la valeur actuelle de la séquence.

```
SQL>select  numdpt_depts.currval
        from sys.dual;
```

14.2.6. Mise en mémoire cache des valeurs d'une séquence

La mise en mémoire cache des séquences permet d'accéder plus rapidement à leurs valeurs. Le cache est chargé dès la première référence à une séquence. Chaque requête demandant une valeur de séquence suivante est extraite de la séquence en mémoire cache. Une fois le dernier numéro utilisé, la requête suivante place d'autres numéros de la séquence en mémoire cache.

Attention aux intervalles

Même si les générateurs de séquences émettent des numéros séquentiels sans « trous », ceux-ci peuvent apparaitre indépendamment d'une validation ou d'une annulation. Par conséquent, si vous annulez un ordre contenant une séquence, le numéro de séquence est perdu. Une défaillance du système peut également générer des interruptions dans une séquence.

Comme les séquences ne sont pas directement liées aux tables, la même séquence peut être utilisée pour plusieurs tables. Dans ce cas, chaque table peut contenir des intervalles entre les numéros de séquence.

Affichage de la prochaine valeur de séquence disponible sans l'incrémenter.

Vous pouvez afficher la prochaine valeur de séquence disponible sans l'incrémenter, à condition que la séquence soit créée à l'aide de NOCACHE lors de l'interrogation de la table USER_SEQUENCES.

```
SQL>select last_number
        from user_sequences;
```

14.2.7. Modification d'une séquence

Si vous atteignez la limite de la valeur maximale (MAXVALUE) de votre séquence, vous ne pouvez plus affecter d'autres valeurs et vous recevez un message d'erreur indiquant que vous avez dépassé la valeur maximale. Pour continuer à utiliser votre séquence, vous pouvez la modifier à l'aide de l'ordre ALTER SEQUENCE.

Syntaxe :

```
ALTER SEQUENCE sequence
        INCREMENT BY n]
         [{maxvalue|nomaxvalue}]
        [{minvalue|nominvalue}]
        [{CYCLE|NOCYCLE}]
        [{CAHE n|NOCACHE}];
```

Exemple :

Modifier la séquence NUMDPT_DEPTS pour passer la valeur de MAXVALUE de 50 à 100

```
SQL>alter sequence numdpt_depts
        maxvalue 100
         nocache
         nocycle;
```

Règles

Pour modifier une séquence

- Vous devez en être le propriétaire ou avoir le privilège ALTER.
- L'ordre ALTER SEQUENCE ne modifie que les numéros de séquence à venir.
- L'option START WITH ne peut pas être modifiée à l'aide de l'ordre ALTER SEQUENCE. Si vous voulez que la séquence commence par un autre numéro, vous devez la supprimer puis la recréer.
- Une validation est exécutée. Par exemple, il n'est pas possible d'imposer une nouvelle valeur maximale (MAXVALUE) qui soit inférieur au numéro de séquence actuelle.

```
SQL>alter sequence numdpt_depts
        maxvalue 5
         nocache
         nocycle;
```

ERREUR à la ligne 1 :

ORA-04009: MAXVALUE ne peut pas être inférieur à la valeur en cours

14.2.8. Suppression d'une séquence

Vous pouvez supprimer une séquence du dictionnaire de données à l'aide de l'ordre DROP SEQUENCE. Pour cela, vous devez être le propriétaire de la séquence ou avoir le privilège DROP ANY SEQUENCE.

Syntaxe :

DROP SEQUENCE *sequence*

où *sequence* représente le nom de la séquence

Exemple :

```
SQL>drop sequence numdpt_depts;
```

14.3. L'index, qu'est-ce que c'est?

Un index Oracle server est un objet de schéma qui permet d'accélérer l'extraction de lignes par le biais d'un pointeur. Il est possible de créer des index, explicitement ou automatiquement. Si la colonne ne comporte pas d'index, la table entière est balayée.

Un index offre un accès direct et rapide aux lignes d'une table. Il a pour but de réduire les E/S sur disque grâce à un chemin d'accès indexé qui identifie rapidement des données. Un index est automatiquement utilisé et mise à jour par Oracle server. Lorsqu'un index est créé, l'utilisateur n'est plus contraint d'intervenir directement.

Les index sont indépendants logiquement et physiquement des tables qu'ils indexent. Autrement dit, vous pouvez à tout moment en créer ou en supprimer, car ils n'ont aucun effet sur les tables de base ou sur d'autres index.

14.3.1. Création d'un index

Il est possible de créer deux types d'index. Le premier, de type unique, est créé automatiquement par Oracle server lorsque vous définissez une colonne de table avec une contrainte PRIMARY KEY ou UNIQUE. Le nom de l'index est celui qui est attribué à la contrainte.

L'autre type d'index, non unique, est créé par l'utilisateur. Par exemple, vous pouvez créer un index sur une colonne définie en FOREIGN KEY, de façon à réduire le temps d'extraction lors d'une jointure.

Syntaxe :

CREATE INDEX *index*
ON *table* (*column*[, *column*]...);

index	représente le nom de l'index
table	représente le nom de la table
column	représente le nom de la colonne de la table à indexer

Exemple :

```
SQL>create index emps_prenom
    on emps (prenom);
```

Visez l'efficacité plutôt que la quantité

Un grand nombre d'index dans une table n'accélère pas forcément l'exécution des requêtes. Chaque opération LMD validée dans une table comportant des index implique que les index doivent être mis à jour. Plus vous associez d'index à une table, plus la mise à jour de tous les index par Oracle server sera longue après une opération LMD.

Quand faut-il créer un index

- Lorsqu'une colonne est fréquemment utilisée dans la clause WHERE ou dans une condition de jointure.
- Lorsqu'une colonne contient un grand nombre de valeurs distinctes.
- Lorsqu'une colonne contient un grand nombre de valeurs NULL.
- Lorsque deux ou plusieurs colonnes sont fréquemment utilisées conjointement dans une clause WHERE ou une condition de jointure.
- Lorsque la table est de grande taille et que la plupart des requêtes doivent extraire moins de 2 à 4% de lignes.

Quand ne faut–il pas créer d'index.

- Lorsque la table est de petite taille.
- Lorsque les colonnes ne sont pas souvent utilisées en tant que condition dans une requête.
- Lorsque la plupart des requêtes sont prévues pour extraire plus de 2 à 4 % de lignes.
- Lorsque la table est fréquemment mise à jour. Si une table comporte un ou plusieurs index, les temps d'accès des ordres LMD à la table sont plus longs, car les index doivent être mise à jour.

14.3.2. Vérification des index

Vérifiez l'existence de vos index à partir de la vue USER_INDEXES du dictionnaire de données.

Vous pouvez également vérifier les colonnes impliquées dans des index en interrogeant la vue USER_IND_COLUMNS.

L'exemple ci-dessous affiche tous les index précédemment créés avec les noms de colonnes et l'unicité de la table EMPS.

```
SQL>select uic.index_name, uic.column_name,
        uic.column_position col_p, ui.uniqueness
        from user_indexes ui, user_ind_columns uic
        where uic.index_name = ui.index_name
        and uic.table_name = 'EMPS';
```

14.3.3. Suppression d'un index

Pour modifier un index, vous devez le supprimer puis le créer. Pour supprimer une définition d'index du dictionnaire de données, utiliser l'ordre DROP INDEX. Pour supprimer un index, vous devez en être le propriétaire ou avoir le privilège DROP ANY INDEX.

Syntaxe :

DROP INDEX *index*

index représente le nom de l'index.

Exemple :

```
SQL>drop index PK_NUMDEPT_DEPTS;
```

15. Les synonymes

Pour vous référer à une table appartenant à un utilisateur, vous devez préfixer le nom de la table par celui de son propriétaire, en séparant les deux par un point. La création d'un synonyme élimine la contrainte de désigner l'objet avec son schéma, en permettant d'attribuer un autre nom à l'objet (table, vue, séquence, procédure etc.). Cette méthode s'avère particulièrement utile pour les noms d'objets longs.

15.1. Création d'un synonyme

Syntaxe :

CREATE [PUBLIC] SYNONYM *synonym*
FOR *object*;

PUBLIC	crée un synonyme accessible à tous les utilisateurs.
synonym	représente le nom du synonyme à créer
object	identifie l'objet pour lequel le synonyme est créé

L'exemple ci-dessous illustre la création d'un synonyme pour la vue DEPTS_SOMM_VUE dans le but d'abréger son nom.

```
SQL>create synonym dsom
      for depts_somm_vue;
```

L'administrateur de base de données peut créer un synonyme public accessible à tous les utilisateurs. Dans l'exemple suivant, le synonyme public *emps* est créé pour la table *emps* de *sownh*.

```
SQL>create public synonym emps
      for sownh.emps;
```

15.2. Suppression d'un synonyme

Pour supprimer un synonyme, utilisez l'ordre DROP SYNONYM. Seul l'administrateur de la base est habilité supprimer un synonyme.

Exemple :

```
SQL>drop public synonym emps
      for sownh.emps;
```

16. Contrôle des accès utilisateurs

16.1. Objectifs

Au terme de ce chapitre, saurez

- Créer des utilisateurs
- Créer des rôles pour faciliter l'installation et la gestion du modèle de sécurité
- Accorder et retirer des privilèges objets.

Dans ce chapitre, vous allez apprendre non seulement à contrôler les accès à des objets spécifiques de la base de données, mais ajouter de nouveaux utilisateurs avec des niveaux différents de privilèges d'accès.

16.2. Contrôle des accès utilisateur

Dans un environnement multi-utilisateur, l'accès et l'utilisation de la base de données doivent être contrôlés à des fins de sécurité. Dans Oracle server, vous pouvez :

- Contrôler l'accès à la base de données
- Autoriser l'accès à des objets spécifique de la base de données
- Confirmer les privilèges accordés reçus dans le dictionnaire de données Oracle
- Créer des synonymes pour les objets de la base de données

La sécurité de la base de données peut être classée en deux catégories : la sécurité du système et la sécurité des données. La sécurité du système couvre l'accès à la base de données et son utilisation au niveau du système (nom d'utilisateur et mot de passe, espace disque alloué aux utilisateurs et opérations système autorisées par l'utilisateur). La sécurité de la base de données couvre l'accès aux objets de la base de données et leur l'utilisation, ainsi que les actions exécutées sur ces objets par les utilisateurs.

16.3. Privilèges

Un privilège donne le droit d'exécuter certains ordres SQL. L'administrateur de base de données est un utilisateur de haut niveau qui est habilité à autoriser les utilisateurs à accéder à la base de données et à ces objets. Les administrateurs doivent obtenir des *privilèges système* pour pouvoir accéder à la base de données, et *des privilèges objets* pour pouvoir manipuler le contenu des objets de la base de données. Ils peuvent aussi avoir reçu le droit d'accorder des privilèges supplémentaires à d'autres utilisateurs ou à des *rôles*, qui sont des groupements de privilèges.

16.4. Schéma

Un *schéma* est une collection d'objets, tels que des tables, des vues et des séquences. Un schéma appartient à un utilisateur de la base de données et porte le même nom que ce dernier.

16.5. Privilèges système

Il existe plus de 80 privilèges système destinés aux utilisateurs et aux rôles. En général, ces privilèges sont accordés par l'administrateur de base de données.

Privilèges types de l'administrateur de base de données

Privilège système	Opérations autorisées
CREATE USER	Permet au bénéficiaire de créer d'autres utilisateurs Oracle (privilège obligatoire pour le rôle d'administrateur de base de données)
DROP USER	Supprime un utilisateur
DROP ANY TABLE	Supprime une table d'un schéma
BACKUP ANY TABLE	Sauvegarde une table dans un schéma à l'aide de l'utilisateur d'export.

16.5.1. Création d'un utilisateur

L'administrateur de base de données créée des utilisateurs en exécutant l'ordre CREATE USER. A ce stade, l'utilisateur n'a pas encore acquis de privilèges. L'administrateur de base de données doit ensuite les lui accorder. Les privilèges déterminent les actions que l'utilisateur peut exécuter au niveau de la base de données.

Syntaxe :

CREATE USER *user*
IDENTIFIED BY *password;*

user	représente le nom de l'utilisateur à créer
password	indique le mot de passe au moyen duquel l'utilisateur doit se connecter

Exemples :

```
SQL>create user olsow
      identified by winde;
```

NB: si le message

```
ERREUR Ó la ligne 1 :
ORA-65096: nom utilisateur ou de rôle commun non valide
```

s'affiche, le contourner par la commande

```
alter session set "_ORACLE_SCRIPT"=true;
```

```
SQL>create user Maimouna
      identified by dalaba;
```

```
SQL>create user Kadiatou
      identified by diaguissa;
```

16.5.2. Privilèges types accordés à un utilisateur

Après avoir créé un utilisateur, l'administrateur de base de données peut lui attribuer des privilèges.

Privilège système	Opérations autorisées
CREATE SESSION	Connexion à la base de données
CREATE TABLE	Création de table dans le schéma de l'utilisateur
CREATE SEQUENCE	Création d'une séquence dans le schéma de l'utilisateur
CREATE VIEW	Création d'une vue dans le schéma de l'utilisateur
CREATE PROCEDURE	Création d'une procédure, d'une fonction ou d'un package stocké dans le schéma de l'utilisateur

16.5.3. Octroi de privilèges système

Syntaxe :

GRANT *privilege* [, *privilege*...]
TO *user* [, *user*...] ;

privilege	Représente le privilège système à accorder
user	Représente le nom de l'utilisateur.

L'ordre GRANT permet à l'administrateur de base de données d'attribuer des privilèges système à un utilisateur. Une fois ces privilèges acquis, l'utilisateur peut immédiatement les exploiter.

Dans l'exemple ci-dessous, les privilèges de création de tables, de séquences, de procédure et de vues sont accordés à l'utilisateur olsow.

```
SQL>grant create table, create sequence, create procedure,
     create view
     to olsow;
```

16.6. Qu'est-ce qu'un Rôle ?

Un rôle est un groupe nommé de privilèges connexes qui peut être accordé à un utilisateur. Cette méthode facilite l'octroi et le retrait des privilèges.

Un utilisateur peut avoir accès à plusieurs rôles, et le même rôle peut être attribué à plusieurs utilisateurs. En général, les rôles sont créés pour une application de base de données.

Création et attribution d'un rôle.

L'administrateur doit d'abord créer le rôle. Ensuite, il attribue des privilèges au rôle, puis le rôle à des utilisateurs.

Création d'un rôle

Syntaxe :

CREATE ROLE *role*;

Où *role* représente le nom du rôle à créer

L'exemple ci-dessus illustre la création du rôle *gestion* autorisant les responsables à créer des tables, des séquences, des procédures et des vues. Ce rôle est ensuite accordé à Oumou, Maimouna et Kadiatou, qui peuvent désormais créer des tables et des vues, des procédures et des séquences.

```
SQL>create role gestion;

SQL>grant create table, create view, create procedure, create
        sequence
        to gestion;

SQL>grant gestion
        to olsow, maimouna, kadiatou;
```

16.7. Modification du mot de passe

Un mot de passe est attribué à chaque utilisateur, que l'administrateur initialise lors de la création de l'utilisateur. Vous pouvez modifier votre mot de passe à l'aide de l'ordre ALTER USER.

Syntaxe :

ALTER USER *user*
IDENTIFIED BY *password*;

user représente le nom de l'utilisateur

password désigne le nouveau mot de passe

En plus de cet ordre, d'autres options permettent de modifier votre mot de passe. Vous devez disposer du privilège ALTER USER pour pouvoir modifier une de ces options.

16.8. Privilèges objet

Un privilège est un droit qui permet d'exécuter des actions sur une table, une vue, une séquence, une procédure, une fonction ou un package spécifique. Les privilèges UPDATE, REFERENCESES et INSERT peuvent être limités et ne s'appliquer qu'à une ou quelques colonnes. Il est possible de limiter le privilège SELECT en créant une vue et en attribuant le privilège SELECT à la vue. Une autorisation associée à un synonyme peut être convertie en une autorisation sur la table de base à laquelle le synonyme fait référence.

Les privilèges objets varient d'un type d'objet à l'autre.

Privilèges objets	Table	Vue	séquence	Procédure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√			

SELECT	√	√	√	
UPDATE	√	√		

16.8.1. Octroi de privilèges objet

Différents privilèges objet sont disponibles pour les différents types d'objets de schéma. Un utilisateur bénéficie automatiquement de tous les privilèges pour les objets de son schéma. En plus, il peut accorder ces privilèges à un autre utilisateur ou à un rôle. Si l'octroi inclut le mot-clé GRANT OPTION, le bénéficiaire peut à son tour accorder le privilège à d'autres utilisateurs.

Syntaxe :

```
GRANT {object_priv [columns] | all}
      ON object
      TO {user | role | PUBLIC}
      [WITH GRANT OPTION];
```

object_priv	représente le privilège objet à accorder
columns	indique les colonnes d'une table ou d'une vue à laquelle des privilèges sont accordés
all	représente tous les privilèges objets
object	représente l'objet auquel les privilèges sont accordés
to	identifie l'utilisateur ou le role auquel le privilège est accordé
public	accorde le privilège objet à tous les utilisateurs
WITH GRANT OPTION	autorise le bénéficiaire à accorder le privilège objet à d'autres utilisateurs et à des rôles

Règles :

- Pour accorder des privilèges sur un objet, ce dernier doit être dans votre propre schéma, ou avoir reçu les privilèges avec le mot-clé WITH GRANT OPTION.
- Le propriétaire d'un objet peut accorder un privilège de cet objet à un autre utilisateur ou à un rôle.
- Le propriétaire d'un objet acquiert automatiquement tous les privilèges de l'objet.

Exemples :

- Accorder le privilège de lecture à *Kadiatou* et *Maimouna* pour leur permettre d'interroger la table *emps*.

```
SQL>grant select
      on emps
      to Kadiatou, Maimouna;
```

- Accorder le privilège UPDATE (pour la mise à jour) à l'utilisateur *Oumou* et au rôle *gestion* sur la table *depts*.

```
SQL>grant update
      on depts
      to oumou, gestion;
```

Remarque:

En général, l'administrateur accorde des privilèges systèmes et les utilisateurs propriétaires des schémas octroient les privilèges objets.

16.8.1.1. Utilisation des mots-clés WITH GRANT OPTION et PUBLIC

- Autoriser un utilisateur à transmettre des privilèges

```
SQL>grant select, insert
      on emps
      to Kadiatou, Maimouna
      with grant option;
```

- Autoriser tous les utilisateurs à interroger les données de la table *emps* de *sownh.*

```
SQL>grant select
      on sownh.emps
      to public;
```

16.8.1.2. Vérification des privilèges accordées

Si vous tentez d'exécuter des opérations non autorisées, Oracle refuse.

Si vous recevez un message d'erreur genre « la vue ou la table n'existe pas », ceci peut être dû à l'une des raisons suivantes :

- Vous avez nommé une table ou une vue inexistante.
- Vous avez tenté d'exécuter une opération sur une table ou une vue pour laquelle vous n'avez pas le privilège approprié.

Vous pouvez accéder au dictionnaire de données pour afficher vos privilèges.

Table du dictionnaire	Description
ROLE_SYS_PRIVS	Privilèges système accordés aux rôles
ROLE_TAB_PRIVS	Privilèges objet accordés aux roles
USER_ROLE_PRIVS	Privilèges objet accessibles par l'utilisateur
USER_TAB_PRIVS_MADE	Privilèges accordés par l'utilisateur
USER_TAB_PRIVS_RECD	Privilèges objet accordés à l'utilisateur
USER_COL_PRIVS_MADE	Privilèges objet liés aux colonnes des objets d'un utilisateur
USER_COL_PRIVS_RECD	Privilèges objet accordés à un utilisateur pour des colonnes spécifiques

16.8.2. Retirer les privilèges objet

On retire les privilèges accordés à d'autres utilisateurs à l'aide de l'ordre REVOKE. Lorsqu'on recourt à cet ordre, les privilèges sont retirés aux utilisateurs que vous nommés ainsi qu'à tous ceux dont ces privilèges ont été accordés par WITH GRANT OPTION.

Syntaxe :

```
REVOKE {privilege [, privilege...] | all}
    ON object
    FROM {user[ ,user...]role|PUBLIC}
    [CASCADE CONSTRAINTS];
```

CASCADE CONSTRAINTS	est obligatoire pour retirer toute contrainte d'intégrité référentielle portant sur l'objet grâce au privilège REFERENCES

Exemple :

Retirer les privilèges SELECT et INSERT accordés aux utilisateurs Kadiatou et Maimouna sur la table emps.

```
SQL>revoke select, insert
    on emps
    from Kadiatou, Maimouna;
```

Table des matières