Cours Oracle SQLP - 1Z0-001

Module n°3
Ordres DML & DDL

ORACLE

Programme de Formation de Supinfo

Laboratoire Supinfo des Technologies Oracle
Auteur : Arnaud Bontemps
Date : 01/01/1601 01:00 - Version 1.2
Nombre de Page : 32
http://www.labo-oracle.com

Ecole Supérieure d'Informatique
23, rue Château Landon
75010 PARIS
http://www.supinfo.com

TABLE DES MATIERES

MODULE N°3 1

ORDRES DML & DDL 1

ORACLE 1

PROGRAMME DE FORMATION DE SUPINFO 1

1 CREATION DES TABLES ET DES CONTRAINTES 4

1.1 Les Tables 4
1.1.1 Introduction 4
1.1.2 Se référer à la table d'un autre utilisateur 4
1.1.3 Conventions de notation 4
1.1.4 L'ordre CREATE TABLE 4

1.2 Type de données et contraintes 5
1.2.1 Types de données Oracle 5
1.2.2 Introduction 6
1.2.3 Types de contrainte 6
1.2.4 Niveaux de définition de contrainte 6

1.3 Définir une Contrainte 6
1.3.1 Contraintes NOT NULL 6
1.3.2 Contraintes UNIQUE 7
1.3.3 Contraintes PRIMARY KEY 7
1.3.4 Contraintes FOREIGN KEY 7
1.3.5 Contraintes CHECK 8

1.4 Tables: Création et vérification 8
1.4.1 Créer une table 8
1.4.2 Créer une tables basée sur une autre table 8
1.4.3 Vérifier la création d'une table 8

2 MANIPULER LES DONNEES 9

2.1 Insérer des données 9
2.1.1 DML et TCS 9
2.1.2 Insérer des lignes dans une table 9
2.1.3 Insérer des valeurs spéciales 10

2.2 Manipulation de données 10
2.2.1 Copier des lignes d'une autre table 10
2.2.2 Modifier des données d'une ou plusieurs lignes 10
2.2.3 Modifier les données de toute les lignes 10
2.2.4 Supprimer des lignes 10

2.3 Processus Transactionnel 11
2.3.1 Introduction sur les transactions 11
2.3.2 Les transactions dans une base de données 11
2.3.3 Valider les modifications 11
2.3.4 Annuler des modifications 12
2.3.5 Annuler des modifications jusqu'à un point de sauvegarde 12

3 MODIFIER DES TABLES ET DES CONTRAINTES 13

3.1 Modifier des colonnes dans une table 13
3.1.1 Ajouter une colonne 13
3.1.2 Modifier une colonne 13
3.1.3 Supprimer une colonne 13

3.2 Modifier les contraintes 14
3.2.1 Ajouter une contrainte 14
3.2.2 Description d'une contrainte 15
3.2.3 Supprimer une contrainte 15
3.2.4 Administrer des contraintes 15

3.3 Modifier une table 16
3.3.1 Supprimer une table 16
3.3.2 Renommer une table 16
3.3.3 Vider le contenu d'une table 16
3.3.4 Ajouter des commentaires pour une table 17

4 LES SEQUENCES 18

4.1 Créer et utiliser les séquences 18
4.1.1 Introduction 18
4.1.2 Créer une séquence 18
4.1.3 Information sur une séquence 19
4.1.4 Utiliser les nombres générés par une séquence 19
4.1.5 Mettre en cache les valeurs d'une séquence 19

4.2 Administrer les séquences 19
4.2.1 Modifier une séquence 19
4.2.2 Supprimer une séquence 20

5 MISE EN PLACE DE VUES 21

5.1 Administrer les vues 21
5.1.1 Introduction 21
5.1.2 L'ordre CREATE VIEW 21
5.1.3 Créer des vues simples 22
5.1.4 Créer des vues complexes 22
5.1.5 Modifier une vue 23
5.1.6 Supprimer une vue 23

5.2 Règles pour les opérations DML 23
5.2.1 Règles pour les opérations DML 23
5.2.2 Règles pour les opérations DML sur des vues 23
5.2.3 La clause WITH CHECK OPTION 23
5.2.4 Empêcher les opérations DML 23
5.2.5 Afficher la définition des vues 24

6 MISE EN PLACE DES SYNONYMES ET INDEX 25

6.1 Création de synonymes 25
6.1.1 Créer un synonyme 25

6.2 Structure d'index 25
6.2.1 Introduction 25
6.2.2 Types d'index 25

6.3 Administrer les index 26
6.3.1 Créer un index 26
6.3.2 Afficher les informations du dictionnaire de données 26
6.3.3 Supprimer un index 27

7 CONTROLE DES ACCES DES UTILISATEURS 28

7.1 Administration des privilèges système 28
7.1.1 Sécurité de la base de données 28
7.1.2 Créer un utilisateur 28
7.1.3 Changer un mot de passe 28
7.1.4 Accorder un privilège 28
7.1.5 Créer un rôle 29

7.2 Administration les privilèges objets 30
7.2.1 Accorder des privilèges objet 30
7.2.2 L'option WITH GRANT OPTION 30
7.2.3 Vérifier les privilège accordés 30
7.2.4 Révoquer un privilège 31

1 CREATION DES TABLES ET DES CONTRAINTES

1.1 Les Tables

1.1.1 Introduction

Une table est un ensemble de lignes et de colonnes. L'intersection d'une ligne et d'une colonne est appelée un champ (ou FIELD).
Une table ne peut contenir que 1000 colonnes et doit être nommée selon les conventions de nommage d'Oracle (cf. 1.1.3).
Une table peut être créée pendant que la base tourne, mais elle sera limitée en taille selon la place disponible sur le serveur (il est tout de même important de prévoir la place que la table va occuper ainsi que son évolution au cours du temps).
De plus il est possible de modifier la structure d'une table, rajouter ou re-définir une colonne pendant que la base est ouverte (le rajout de lignes se fait avec la commande INSERT).

1.1.2 Se référer à la table d'un autre utilisateur

Il existe sous Oracle un objet qui regroupe tous les objets appartenant à un utilisateur : le schéma. Une table appartenant à un schéma ne peut appartenir à un autre schéma.
Il faut alors pour accéder aux objets d'un autre utilisateur que celui-ci autorise l'accès en donnant les droits nécessaires. Il sera ensuite possible d'y faire référence en utilisant la syntaxe suivante :

nom_du_schema**.**nom_de_la_table

Exemple :

```
SCOTT.EMP
```

1.1.3 Conventions de notation

Il existe 4 règles pour nommer les objets d'Oracle :

- L'identifiant doit commencer par une lettre et peut contenir un maximum de 30 caractères.
- L'identifiant ne pourra contenir que les caractères suivant :
  A à **Z**, **a** à **z**, **0** à **9**, **_** , **$**, **#** et pas le "**&**"
- L'identifiant ne peut pas être identique à celui d'un autre objet déjà présent dans le schéma.
- L'identifiant ne devra pas correspondre à un mot Oracle réservé tels que Table, Key, Create, Unique, Constraint…

En plus de ces 4 règles, il existe quelques lignes de conduite à respecter :

- Le nom de la table et des colonnes doit être explicites et représentatifs des informations contenues dans la table et les colonnes.
- Il est fortement conseillé de nommer de manière identique une colonne qui fait référence à une autre colonne d'une autre table (cf. : 1.3.3 et 1.3.4)

L'identifiant n'est pas sensible à la casse. Oracle permet de contourner les règles de nommage en utilisant la notation suivante pour les noms de tables ou de colonne :

"nom_de_l'identifiant**"**

Cette notation permet d'utiliser les mots réservés et d'être sensible à la casse (il est fortement déconseillé d'utiliser cette notation pour éviter toutes erreurs lors de futures requêtes.)

1.1.4 L'ordre CREATE TABLE

Pour créer une table il est nécessaire de posséder le privilège CREATE TABLE et d'utiliser la syntaxe de création d'une table:

CREATE TABLE [schema.] nom_table (
```
column_name datatype [ default expr ]
[ column_constraint ], …
[ table_constraint ], … ) ;
```

datatype	Défini le type de données de la colonne ainsi que sa taille.
default expr	Défini la valeur par défaut de la colonne. Cette valeur peut être littérale, une expression ou bien une fonction SQL.
column_constraint	Défini une contrainte d'intégrité sur les données entrées dans la colonne.
table_constraint	Défini des contraintes d'intégrité portant sur plus d'une colonne.

1.2 Type de données et contraintes

1.2.1 Types de données Oracle

Il existe 10 types de données :

Type de données	***Description***
VARCHAR2 (taille)	Texte de longueur variable de longueur minimum de 1 caractère et pouvant aller jusqu'à 4000 caractères. La taille devra obligatoirement être définie.
CHAR (taille)	Texte de longueur fixe de taille minimum de 1 caractère et pouvant aller jusqu'à 2000 caractères.
NUMBER (p, e)	Nombre ayant une précision pouvant aller de 1 à 38 chiffres et ayant une échelle pouvant aller de –84 à 127. (L'échelle est en fait le nombre de chiffre à afficher après la virgule.)
DATE	Date notée sous la forme DD-MON-YY.
LONG	Texte de longueur variable pouvant stocker jusqu'à 2 gigas. On ne pourra mettre qu'une colonne de type LONG par table.
RAW (taille)	Equivalent à VARCHAR2, mais il permet de stocker des données binaires qui ne sont pas interprétées par Oracle. La taille maximum est de 2000.
LONGRAW	Equivalent à LONG mais pour des données de type binaire non interprétées par Oracle.
CLOB	Permet de stocker un pointeur vers un fichier de données composé de caractère et pouvant contenir jusqu'à 4 gigas.
BLOB	Permet de stocker un pointeur vers un fichier composé de données binaire et pouvant contenir jusqu'à 4 gigas.
BFILE	Permet de stocker les données binaires d'un fichier externe pouvant contenir jusqu'à 4 gigas.

1.2.2 Introduction

Oracle utilise les contraintes afin d'éviter à l'utilisateur de saisir des données erronées dans une table.
Les contraintes permettent aussi de vérifier les opérations d'insertions et d'effacements qui doivent satisfaire les contraintes existantes.
Les contraintes peuvent être rajoutées ou enlevées après la création de la table. Les contraintes de clé permettent aussi de faire référencer une colonne (ex: Foreign Key) à une autre colonne (Primary Key). On ne pourra pas assigner de contraintes à un type de données.

1.2.3 Types de contrainte

Il existe 5 types de contraintes :

NOT NULL	Permet d'empêcher de rajouter des valeurs nulles dans la colonne, (c'est la seule contrainte à être affichée par la commande DESCRIBE.)
UNIQUE	Permet de vérifier que les valeurs entrées ne sont pas déjà présentes dans la colonne (même si la valeur est nulle). Cette contrainte permet d'être sur d'avoir des valeurs uniques dans la colonne. Une colonne UNIQUE accepte les valeurs nulles. De plus Oracle crée automatiquement un index pour cette colonne.
PRIMARY KEY	Est définie pour une colonne ou un ensemble de colonnes qui contiendront un identifiant unique et non nul et qui permettra d'identifier de manière unique un enregistrement (ou ligne) de la table. La contrainte PRIMARY KEY combine les contraintes NOT NULL et UNIQUE. Il ne peut y avoir qu'une seule PRIMARY KEY par table.
FOREIGN KEY	Est définie pour une colonne ou une combinaison de colonnes qui font références à une autre colonne Primary Key dans une table établissant ainsi une relation entre les deux tables. Les valeurs de la Foreign Key devront correspondre aux valeurs de la colonne Primary Key à laquelle la Foreign Key fait référence ou alors être nulle.
CHECK	Permet de définir une condition que la colonne doit satisfaire (ex : le numéro de département devra être compris entre 10 et 99). Il n'y pas de limite sur le nombre de ce type de contraintes. Elles ne peuvent pas utiliser les commandes : SYSDATE, UID, USER, USERENV, ou faire des appels sur des lignes.

1.2.4 Niveaux de définition de contrainte

Il existe deux types de niveau pour des contraintes :

- Contrainte de niveau de colonne : contrainte basée sur une colonne pouvant être définie au niveau de la définition de la colonne ou
- Contrainte de niveau de table : contrainte basée sur une ou plusieurs colonnes et est définie après les définitions de colonnes.

La seule contrainte qui devra être obligatoirement définie au niveau de la colonne est la contrainte NOT NULL.
Toutes les informations sont disponible dans les vues USER_CONSTRAINTS.
Il est préférable de nommer soit même les contraintes lors de leur création. Si les contraintes ne sont pas nommées, Oracle sans charge automatiquement. Et dans ce cas, les noms des contraintes ne sont ni représentatifs ni explicites. Il est donc plus difficile de les retrouver et de les manipuler.

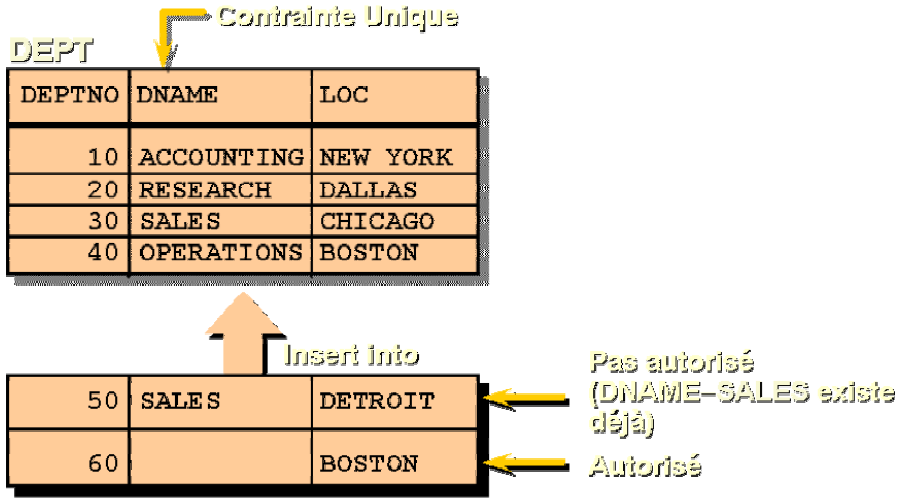
1.3 Définir une Contrainte

1.3.1 Contraintes NOT NULL

Exemple :

```
SQL> CREATE TABLE dept
  2  (…
  3  deptno NUMBER(2) CONSTRAINT dept_deptno_nn NOT NULL,
  4  …);
```

1.3.2 Contraintes UNIQUE



Exemple :

```
SQL> CREATE TABLE emp
  2  (empno NUMBER(4) CONSTRAINT emp_empno_uk UNIQUE,
  3  …
  4  ename VARCHAR2(10));
```

Si la contrainte UNIQUE s'applique à plusieurs colonnes il est possible de les mettre entre parenthèses et de les séparer par des virgules.

1.3.3 Contraintes PRIMARY KEY

Exemple :

```
SQL> CREATE TABLE dept
  2  (dname VARCHAR2(14),
  3  LOC VARCHAR2(13),
  4  Deptno NUMBER(2) CONSTRAINT dept_deptno_pk PRIMARY KEY);
```

```
SQL> CREATE TABLE lili
  2  (id1 NUMBER(2),
  3  id2 NUMBER(3),
  4  CONSTRAINT id1_id2_lili_pk PRIMARY KEY (id1, id2));
```

Dans le deuxième exemple la clé primaire sera basée sur les colonnes id1 et id2.

1.3.4 Contraintes FOREIGN KEY

Exemple :

```
SQL> CREATE TABLE emp
  2  (ename VARCHAR2(10),
  3  empno NUMBER(4) NOT NULL,
  4  deptno NUMBER(2) CONSTRAINT emp_deptno_fk
  5  REFERENCES dept(deptno));
```

```
SQL> CREATE TABLE wean
```

```
  2  (id3 NUMBER(2) CONSTRAINT id3_wean_pk PRIMARY KEY,
  3  fk1 NUMBER(2), fk2 NUMBER(2),
  4  CONSTRAINT fk1_fk2_wean_fk FOREIGN KEY (fk1, fk2)
  5  REFERENCES lili(id1, id2));
```

1.3.5 Contraintes CHECK

Exemple :

```
SQL> CREATE TABLE ord (
  2           orderdate DATE,
  3           shipdate DATE,
  4  CHECK (orderdate<=shipdate));
```

1.4 Tables: Création et vérification

1.4.1 Créer une table

Exemples :

```
SQL> CREATE TABLE dept
  2  (deptno NUMBER(2)
  3           CONSTRAINT dept_deptno_pk PRIMARY KEY,
  4  dname VARCHAR2(25)
  5           CONSTRAINT dept_dname_nn NOT NULL);
```

```
SQL> CREATE TABLE employee
  2  (sid NUMBER, name VARCHAR2(50), last_name VARCHAR2(50));
```

```
SQL> CREATE TABLE dept
  2  (deptno NUMBER(2), dname VARCHAR2(14), loc VARCHAR2(13));
```

1.4.2 Créer une tables basée sur une autre table

CREATE TABLE nom_table AS ordre_select copie la structure et les données issues de l'ordre SELECT mais ne copie pas les contraintes.

Exemple :

```
SQL> CREATE TABLE newemp AS
  2           SELECT    empno,deptno
  3           FROM      emp
  4           WHERE     deptno=10;
```

1.4.3 Vérifier la création d'une table

Pour vérifier que la table à bien été crée il est possible d'utiliser la commande DESCRIBE qui permet d'afficher les colonnes d'une table avec leur type, leur taille et leur contrainte NOT NULL.
Il est aussi possible d'aller chercher d'autres informations dans les vues USER_TABLES, USER_OBJECTS et USER_CATALOG.

2 MANIPULER LES DONNEES

2.1 Insérer des données

2.1.1 DML et TCS

Les actions de modification, de rajout, de suppression des données dans une base de données exécutent un ordre SQL de type DML (**D**ata **M**anipulation **L**anguage).

Il y a 3 ordres DML :

- INSERT : Pour insérer des données dans une table
- UPDATE : Pour modifier des données dans une table
- DELETE : Pour supprimer des données dans une table

Pour exécuter ces trois ordres SQL, l'utilisateur doit posséder les privilèges INSERT, UPDATE et DELETE sur un objet. De plus si les tables ne sont pas dans son schéma, le propriétaire des tables ou un administrateur devra donner les droits nécessaires pour pouvoir effectuer des opérations sur ces tables.
Le résultat de ces ordres DML est contrôlé par une transaction (= collection d'ordre DML). Une transaction peut être administrée par un groupe d'ordres de contrôle de transaction (**T**ransaction **C**ontrol **S**tatement).

Il y a 3 ordres TCS :

- COMMIT :
  Valide tout les ordres DML en attente
- SAVEPOINT :
  Permet de fixer des points de sauvegarde qui permettront de revenir en arrière si une erreur se produit grâce à la commande ROLLBACK. Cette commande ne fait pas partie de la norme SQL Standard AINSI
- ROLLBACK :
  Permet de revenir à l'état avant les transactions DML mais permet de revenir aussi à un SAVEPOINT.

2.1.2 Insérer des lignes dans une table

Syntaxe de l'ordre INSERT :

```
INSERT INTO   nom_de_table
                   [ ( column_name [, column_name ] …) ]
VALUES             ( value [, value …] );
```

[(column_name [, column_name] …)]	Permet de définir les colonnes dans lesquelles les valeurs seront insérées. Si l'insertion se fait dans toutes les colonnes cette clause sera retirée de l'ordre INSERT.
VALUES (value [,value …]);	Permet de définir les valeurs à rajouter dans les colonnes de la table.

Exemple:

```
SQL> INSERT INTO EMP (empno, ename, hiredate)
  2  VALUES (4242, 'DUPOND', '21/08/01');
```

Si aucunes valeurs n'est spécifiées lors de l'insertion dans une colonne NULL, Oracle insérera automatiquement une valeur nulle. Si la colonne possède la contrainte NOT NULL, Oracle affichera alors une erreur. Il n'est possible d'ajouter qu'une seule ligne à la fois avec la clause VALUES.

2.1.3 Insérer des valeurs spéciales

Oracle fournit la possibilité d'insérer la date du jour avec la fonction SYSDATE et le nom de l'utilisateur courant avec la fonction USER.

2.2 Manipulation de données

2.2.1 Copier des lignes d'une autre table

Il est possible grâce à l'ordre INSERT de copier les valeurs d'une table dans une autre sans avoir à en saisir les données. Pour cela il suffit d'utiliser une requête SQL pour aller chercher les valeurs automatiquement.

```
INSERT INTO nom_table [(col1[,col2,…])]
SELECT …;
```

Exemple :

```
SQL> INSERT INTO managers (id, name, salary, hiredate)
  2  SELECT empno, ename, sal, hiredate
  3  FROM emp
  4  WHERE job='MANAGER';
```

Attention : La clause VALUES ne doit pas être utilisée dans ce type de requête. De plus le nombre de colonne passée dans la clause INTO doit correspondre au nombre de colonnes sélectionnées dans la requête SELECT.

2.2.2 Modifier des données d'une ou plusieurs lignes

Il est possible grâce à l'ordre UPDATE de modifier des valeurs de colonnes dans les tables.
Voici la syntaxe de l'ordre UPDATE :

```
UPDATE        nom_table
SET           column_name= value [, column_name = value ]
[ WHERE       condition ]
```

Exemple:

```
SQL> UPDATE emp
  2  SET comm=1000
  3  WHERE ename='DUPOND';
```

La clause SET permet de spécifier les colonnes à modifiées avec leur nouvelles valeurs, la clause WHERE permet de définir les critères de sélection des lignes ou enregistrements à modifier.
Il est conseillé d'utiliser la clé primaire de la table afin de sélectionner de manière efficace le champ souhaité.

2.2.3 Modifier les données de toute les lignes

On peut aussi modifier toutes les lignes d'une table en omettant la clause WHERE.

Exemple:

```
SQL> UPDATE sal
  2  SET sal=sal+50;
```

2.2.4 Supprimer des lignes

Pour effacer une ou plusieurs lignes d'une table il suffira d'utiliser la commande DELETE dont voici la syntaxe :

```
DELETE        [FROM] table
```

```
WHERE          condititon;
```

Exemple:

```
SQL> DELETE FROM emp
  2  WHERE ename='DUPOND';
```

Dans le cas où l'on efface une ligne qui contient une Primary Key, Oracle vérifie automatiquement que celle ci n'est plus utilisée dans une table fille et si c'est le cas renvoie une erreur. (Il ne faut pas oublier que si une erreur se produit, 'il est toujours possible de recourir aux ordres TCS.)

2.3 Processus Transactionnel

2.3.1 Introduction sur les transactions

Les ordres TCS donnent plus de facilité pour gérer toutes les transactions et garantissent la consistance des données lorsque le processus de l'utilisateur plante ou lorsqu'il y a une coupure de courant.

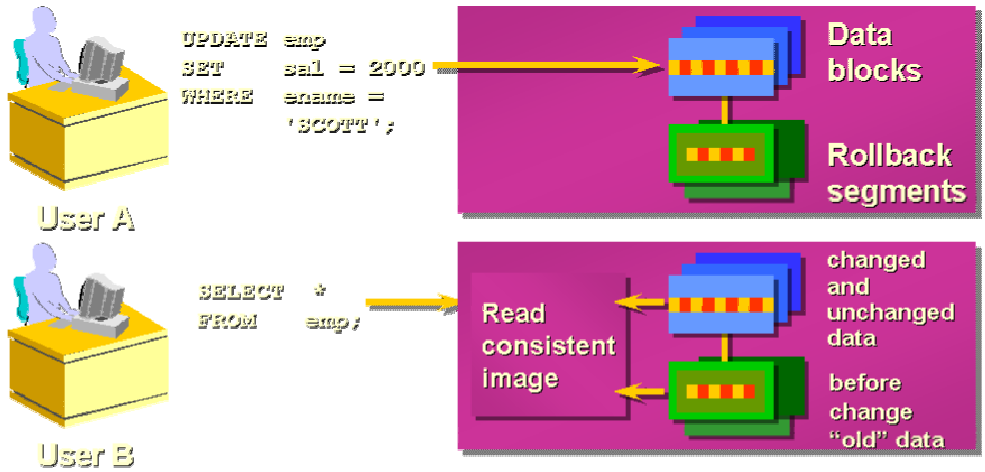
2.3.2 Les transactions dans une base de données

Une transaction commence dès que la base de données rencontre le 1er ordre SQL exécutable. Le second ordre SQL exécutable rencontré démarrera automatiquement la seconde transaction. Il y a différentes actions possible à la fin d'une transaction comme COMMIT et ROLLBACK.
Certains ordres SQL du type DDL (**D**ata **D**efinition **L**anguage) ou du type DCL (**D**ata **C**ontrol **L**anguage) sont "auto-commité" et terminent donc toutes les transactions en cours.
Quitter SQL*Plus termine aussi les transactions en cours. Si la machine plante les transactions sont annulées.
Vous pouvez donc valider ou non à n'importe quel moment les transactions en cours.

2.3.3 Valider les modifications

La syntaxe de la commande COMMIT est la suivante :

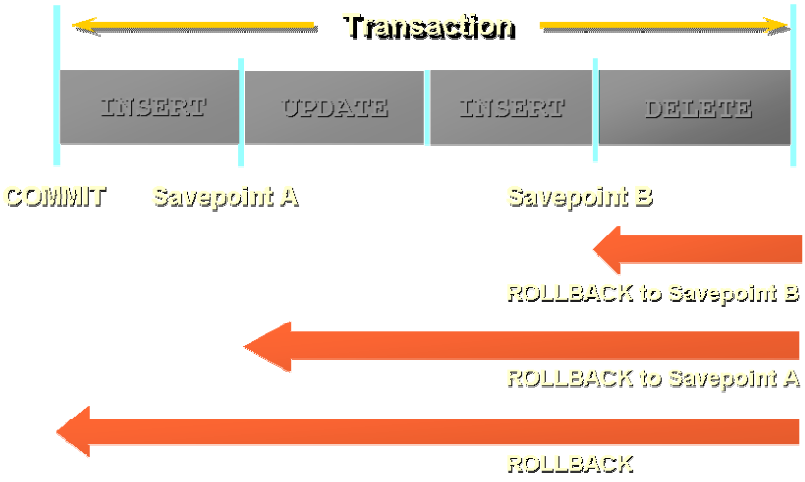
```
SQL> COMMIT;
```



Tant que vous n'avez pas validez des modifications, Oracle garde en mémoire une image avant (modifications) qui permet aux autres utilisateurs de pouvoir continuer à travailler et d'exécuter des ordres SELECT sur la table "avant" que les modifications ne soient intervenues.
Si un autre utilisateur tente de faire une modification sur les lignes que vous modifiez il devra alors attendre que vous ayez fini et validez ou annuler votre transaction pour pouvoir faire les siennes.
Une transaction pose un verrou sur toutes les données qui vont être modifiées afin d'éviter les redondances. A la fin de la transaction les verrous sont retirés et les SAVEPOINTS effacés.

2.3.4 Annuler des modifications

Supposons que vous ayez fait une erreur lors de votre transaction, vous pouvez l'annuler grâce à la commande ROLLBACK qui restitue en fait l'image avant sur le serveur. La commande ROLLBACK termine la transaction et enlève tout les verrous.



Exemple :

```
SQL> ROLLBACK ;
```

2.3.5 Annuler des modifications jusqu'à un point de sauvegarde

Il peut arriver qu'il ne soit pas nécessaire d'annuler toutes les modifications effectuées. Pour cela il est possible de jalonner vos modifications de marqueurs de sauvegarde SAVEPOINT. Ceux ci permettront de faire des ROLLBACKS "réduits".
La syntaxe de la commande SAVEPOINT est la suivante :

```
SAVEPOINT nom;
```

Exemple:

```
SQL> SAVEPOINT save_a;
```

Il est possible d'utiliser le même nom pour tous les points de sauvegarde, dans ce cas Oracle ne gardera que le dernier portant ce nom.
La syntaxe pour revenir à un point de sauvegarde est la suivante :

```
ROLLBACK TO nom_du_savepoint;
```

Exemple:

```
SQL> ROLLBACK TO save_a;
```

3 MODIFIER DES TABLES ET DES CONTRAINTES

Il est parfois nécessaire de modifier la structure d'une table pour suivre l'évolution de l'entreprise.

3.1 Modifier des colonnes dans une table

Pour modifier une table il faudra employer la commande : ALTER TABLE nom_de_la_table
ATTENTION il n'est possible de supprimer une colonne qu'avec un **Oracle 8I**.

3.1.1 Ajouter une colonne

Voici la syntaxe pour ajouter une colonne à une table :

```
ALTER TABLE   nom_table
ADD                     (column datatype
                        [DEFAULT expr]
                        [NOT NULL]
                        [,column datatype]…);
```

Exemple:

```
SQL> ALTER TABLE employee
  2  ADD (birth DATE NOT NULL);
```

Touts les paramètres rencontrés sont identiques à ceux de la commande CREATE TABLE (cf 1.4.1). De plus il n'est pas possible de spécifier la position d'insertion d'une colonne. Celle-ci est insérée directement à la suite des autres.
Si les colonnes qui étaient déjà présente dans la table contenaient des valeurs alors la nouvelle colonne sera initialisée à NULL pour toutes les lignes.

3.1.2 Modifier une colonne

Il existe 4 modifications de colonnes possibles :

- Modifier son type de données :
  Il n'est possible de modifier le type de données d'une colonne que si la table ne contient aucune lignes ou si la colonne ne contient que des valeurs nulles.
- Modifier sa taille :
  Il n'est possible de modifier la taille d'une colonne que si la table ne contient aucune lignes ou si la colonne ne contient que des valeurs nulles.
- Modifier sa valeur par défaut:
  Cette modification sert à modifier la valeur par défaut de la colonne pour les valeurs à venir.
- Ajouter ou enlever sa contrainte NOT NULL:
  Il n'est possible d'ajouter cette contrainte seulement si la colonne est vide ou ne contient aucune valeur nulle.

Pour modifier les caractéristiques d'une colonne on utilisera la syntaxe suivante :

```
ALTER TABLE   nom_table
MODIFY                  (column datatype
                        [DEFAULT expr]
                        [,column datatype] …);
```

Exemple:

```
SQL> ALTER TABLE employee
  2  MODIFY (birth DATE DEFAULT '01/01/01');
```

3.1.3 Supprimer une colonne

Voici la syntaxe utilisée pour supprimer une colonne sous **Oracle 8I** :

```
ALTER TABLE   nom_table
DROP COLUMN   nom_colonne
[CASCADE CONSTRAINTS]
[INVALIDATE];
```

Exemple:

```
SQL> ALTER TABLE employee
  2  DROP COLUMN birth
  3  CASACADE CONSTRAINTS
  4  INVALIDATE;
```

Il n'est possible de supprimer seulement qu'une colonne à la fois. Cette colonne pourra ou non contenir des données. De plus il n'est possible de supprimer une colonne que si la table contient encore au moins une colonne après la suppression de la colonne.

La commande CASCADE CONSTRAINTS permet d'effacer aussi les contraintes d'intégrités liées à la colonne effacée.
La commande INVALIDATE permet aussi d'invalider touts les objets qui faisait références à cette colonne. Un objet invalidé sera automatiquement validé la prochaine fois qu'un utilisateur y fera référence.

Il est possible de définir des colonnes qui ne sont pas régulièrement utilisées comme étant des colonnes UNUSED grâce à la syntaxe suivante :

```
ALTER TABLE   nom_table
SET UNUSED        nom_colonne;
```

Exemple:

```
SQL> ALTER TABLE employee
  2  SET UNSED last_name;
```

Ceci permet ensuite de supprimer toutes les colonnes inutilisées en une seule fois grâce à la syntaxe :

```
ALTER TABLE   nom_table
DROP UNUSED COLUMNS;
```

Exemple:

```
SQL> ALTER TABLE employee
  2  DROP UNSED COLUMNS;
```

Une fois qu'une colonne est UNUSED, un ordre SELECT * ne rapportera pas les données de cette colonne.

3.2 Modifier les contraintes

Il est possible d'ajouter, supprimer, activer ou de désactiver une contrainte mais il n'est pas possible de modifier sa structure.

3.2.1 Ajouter une contrainte

Voici la syntaxe pour ajouter une contrainte d'intégrité à une colonne :

```
ALTER TABLE   nom_table
ADD CONSTRAINT    nom_contrainte
                      Constraint_type (nom_column);
```

Exemple:

```
SQL> ALTER TABLE employee
  2  ADD CONSTRAINT employee_sid_pk
  3  PRIMARY KEY (sid);
```

Comme vu précédemment, il est recommandé de nommer par soit même les contraintes pour des facilités de déboguage.
Les différents types de contraintes sont les suivants :

- UNIQUE
- CHECK
- PRIMARY KEY
- FOREIGN KEY

Il n'est pas possible de rajouter la contrainte NOT NULL avec la commande ADD CONSTRAINT.

3.2.2 Description d'une contrainte

En utilisant la commande DESCRIBE sur une table la seule contrainte qu'il est possible de voir apparaître est la contrainte NOT NULL. Pour pouvoir connaître les autres contraintes présentes sur une table, il est nécessaire d'aller chercher ces informations dans les vues système USER_CONSTRAINTS et USER_CONS_COLUMNS.

3.2.3 Supprimer une contrainte

Voici la syntaxe pour supprimer une contrainte :

```
ALTER TABLE   nom_table
DROP CONSTRAINT   nom_contrainte
[CASCADE];
```

Exemple:

```
SQL> ALTER TABLE employee
  2  DROP CONSTRAINT employee_sid_pk
  3  CASCADE;
```

S'il est nécessaire de supprimer la clé primaire de la table il suffira de préciser à la place du nom de la contrainte PRIMARY KEY. La commande CASCADE vous permettra de supprimer les contraintes FK automatiquement dans les tables filles.

3.2.4 Administrer des contraintes

Il peut parfois être utile de désactiver certaines contraintes pendant une courte durée puis de les réactiver ensuite.
Voici la syntaxe permettant de désactiver une contrainte:

```
ALTER TABLE   nom_table
DISABLE [VALIDATE|NOVALIDATE] CONSTRAINT nom_contrainte
[CASCADE];
```

Exemple:

```
SQL> ALTER TABLE employee
  2  DISABLE NOVALIDATE CONSTRAINT employee_sid_pk
  3  CASCADE;
```

Lors de l'utilisation de la commande CASCADE sur une PRIMARY KEY toute les FOREIGN KEY qui y font références sont automatiquement désactivées mais ne seront pas réactivées automatiquement lors de la réactivation de la PK.
Voici la syntaxe qui va vous permettre de réactiver une contrainte :

```
ALTER TABLE   nom_table
ENABLED [VALIDATE|NOVALIDATE] CONSTRAINT nom_contrainte;
```

Exemple:

```
SQL> ALTER TABLE employee
  2  ENABLE VALIDATE CONSTRAINT employee_sid_pk;
```

La clause VALIDATE permet sous Oracle 8I de vérifier que la contrainte est bien appliquée sur toutes les nouvelles et anciennes valeurs de la colonne. La clause NOVALIDATE ne vérifiera pas les anciens enregistrements mais seulement les nouveaux.

3.3 Modifier une table

3.3.1 Supprimer une table

Il va parfois falloir supprimer une table avec ses données et tout ses index.
Voici la syntaxe pour supprimer cette table :

```
DROP TABLE        nom_table
[CASCADE CONSTRAINTS];
```

Exemple:

```
SQL> DROP TABLE employee
  2  CASCADE CONSTRAINTS;
```

CASCADE CONSTRAINTS permet de supprimer de manière automatiquement toutes les contraintes d'intégrité de cette table.
Seul le propriétaire ou un utilisateur ayant le privilège DROP ANY TABLE pourra supprimer une table. De plus après la suppression de la table toutes les vues et synonymes qui y faisaient références restent en place et sont invalidés.

Attention DROP TABLE étant un ordre DDL il est auto-commité et valide par conséquent toutes les transactions en attente.

3.3.2 Renommer une table

Voici la syntaxe pour renommer une table :

```
RENAME        ancien_nom_table
TO            nouveau_nom_table;
```

Exemple:

```
SQL> RENAME employee TO emp_table;
```

Seul le propriétaire de la table pourra la renommer.

3.3.3 Vider le contenu d'une table

Les commandes TRUNCATE et DELETE permettrent de vider une table de toutes ses données tout en conservant sa structure :

```
TRUNCATE TABLE    nom_table;
DELETE        nom_table;
```

Exemple:

```
SQL> TRUNCATE employee;
```

```
SQL> DELETE employee;
```

Attention TRUNCATE est un ordre DDL et est donc auto-commité, après son utilisation il faudra désactiver tout les Foreign Key présentent sur la table.
Seul le propriétaire ou un utilisateur ayant le privilège DELETE ANY TABLE ou DELETE pourra vider une table.

La différence entre TRUNCATE et DELETE est que TRUNCATE ne génère pas d'informations de ROLLBACK et est donc, par conséquent, plus rapide.

3.3.4 Ajouter des commentaires pour une table

Il est possible de rajouter des commentaires sur les tables dans le dictionnaire de donnée en utilisant la commande COMMENT. Cela permet de documenter les objets et de mieux comprendre leur fonction.
Il est possible de consulter les commentaires dans les vues système suivantes:

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

Il est possible de saisir jusqu'à 2000 caractères en utilisant la syntaxe suivante:

```
COMMENT ON TABLE  nom_table
[COLUMN                        table.column]
IS                             'texte';
```

Exemple:

```
SQL> COMMENT ON TABLE EMP
  2  IS 'Table des employés.';
```

Le paramètre " COLUMN" est optionnel et servira juste à préciser sur quelle colonne porte le commentaire. S'il n'est pas spécifié, le commentaire portera sur la table.
Pour supprimer un commentaire on en créera un vide (c.a.d en remplaçant le texte par ' ').

4 LES SEQUENCES

Une séquence est un objet de la base de données qui génère des numéros uniques qui pourront être insérés dans une colonne (comme par exemple une Primary Key).

4.1 Créer et utiliser les séquences

4.1.1 Introduction

Les séquences ne sont pas stockées avec les tables et peuvent donc être utilisées par tous les utilisateurs. Une séquence incrémente ou décrémente un compteur dans une routine Oracle (ce qui génère un gain de temps et de codage). Il est possible de stocker les valeurs dans un cache, ce qui accélèrera l'accès au données de la séquence. Pour créer des séquences il est nécessaire de posséder le privilège CREATE SEQUENCE

4.1.2 Créer une séquence

Voici la syntaxe de la commande qui permet de créer une séquence :

```
CREATE SEQUENCE nom_sequence
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE|NOCYCLE]
[CACHE n | NOCACHE];
```

INCREMENT BY	Incrémentation de la colonne (valeur par défaut : 1)
START WITH	Départ du compteur (valeur par défaut : 1)
MAXVALUE | NOMAXVALUE	Valeur maximale du compteur ou pas de valeur maximale. La valeur maximale sera de 10^27 pour une incrémentation et de –1 pour une décrémentation.
MINVALUE|NOMINVALUE	Valeur minimale du compteur ou pas de valeur minimale. La valeur minimale sera de 1 pour une séquence ascendante et de – (10^26) pour une séquence descendante.
CYCLE|NOCYCLE	Autorise la séquence à reprendre le compteur à partir de START WITH une fois qu'elle a atteint MAXVALUE ou MINVALUE. A ne pas utiliser dans le cas d'une utilisation pour une colonne PK.
CACHE n | NOCACHE	Met ou non des valeurs en cache (valeur par défaut : 20).

Le nom d'une séquence est composé du nom de la table, à laquelle appartient la colonne qu'elle incrémente, et le nom de la colonne qu'elle incrémente. Cette convention de nommage permet de retrouver et identifier facilement les séquences.

Exemple :

Une séquence qui incrémente la colonne empno de la table emp devra s'appeler emp_empno.

```
SQL> CREATE SEQUENCE emp_empno
  2  INCREMENT BY 1
  3  START WITH 1
  4  MAXVALUE 9999
  5  MINVALUE 0
  6  NOCYCLE
  7  CACHE 25;
```

4.1.3 Information sur une séquence

Il est possible de voir vos objets dans la vue système USER_OBJECTS qui contient le nom, le type, la date de création, le statut de l'objet. Mais pour avoir des informations plus précise sur les séquences, il suffira de regarder dans la vue système USER_SEQUENCES.

4.1.4 Utiliser les nombres générés par une séquence

Pour récupérer les valeurs d'une séquence il faudra utiliser les pseudo colonnes: NEXTVAL et CURRVAL.

Remarque : Attention il est possible de les sélectionner mais pas de faire d'insertion de modification ou d'effacement dans ces pseudo colonnes.

Pour récupérer la valeur actuelle d'une séquence il suffira de faire appel à la pseudo colonne CURRVAL de la manière suivante : nom_sequence.CURRVAL.

Pour récupérer la valeur suivante d'une séquence il suffira de faire appel à la pseudo colonne NEXTVAL de la manière suivante : nom_sequence.NEXTVAL

NEXTVAL stocke de manière automatique la valeur suivante de la séquence dans CURRVAL.
NEXTVAL renverra une valeur unique à chaque référence à la séquence. Chaque valeur sera unique quel que soit l'utilisateur qui y fait référence.

Exemple :

Si l'utilisateur A fait appel à NEXTVAL alors CURVAL=2
Si l'utilisateur B fait appel à NEXTVAL alors CURVAL=3

De plus une séquence ne revient jamais en arrière même si des ROLLBACK sont exécutés. Les valeurs utilisées ne seront pas remises dans le cache. De même si le système plante, les valeurs, stockées en mémoire, seront alors perdues.

Exemple :

Utilisation d'une séquence dans un ordre INSERT

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (dept_deptno.NEXTVAL, 'MARKETING', 'SAN DIEGO');
```

4.1.5 Mettre en cache les valeurs d'une séquence

Il peut être intéressant de mettre en cache des valeurs d'une séquence. La première fois qu'un ordre ou une commande fait référence à la procédure, celle-ci remplit alors le cache de valeurs et chaque fois qu'un utilisateur fait référence à la séquence pour obtenir une valeur, celle-ci est tirée du cache jusqu'à ce qu'il n'y en ai plus et que la séquence remplisse de nouveau celui-ci.

4.2 Administrer les séquences

4.2.1 Modifier une séquence

Il peut parfois être intéressant de modifier les caractéristiques d'une séquence, comme sa valeur maximale (MAXVALUE). Pour cela il suffira de faire appel à la commande ALTER SEQUENCE qui affectera uniquement les nouvelles valeurs à venir.

```
ALTER SEQUENCE nom_sequence
[ INCREMENT BY n ]
[ { MAXVALUE n | NOMAXVALUE } ]
[ { MINVALUE n | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ]
[ { CACHE n | NOCACHE } ] ;
```

Exemple:

```
SQL> ALTER SEQUENCE emp_empno
  2  INCREMENT BY 2
  3  MAXVALUE 7777
  4  CYCLE
  5  CACHE 30;
```

Il n'est pas possible de changer sa valeur de départ (START WITH) grâce à la commande ALTER SEQUENCE. Le seul moyen sera d'effacer la séquence et de la recréer.
Après les modifications, Oracle vérifiera alors que celles-ci sont valides (nouveau MAXVALUE supérieur à la valeur actuelle…)

4.2.2 Supprimer une séquence

Pour pouvoir supprimer une séquence de son propre schéma ou de n'importe quel schéma, il faut posséder les privilèges DROP SEQUENCE ou DROP ANY SEQUENCE:

```
DROP SEQUENCE nom_sequence;
```

Exemple:

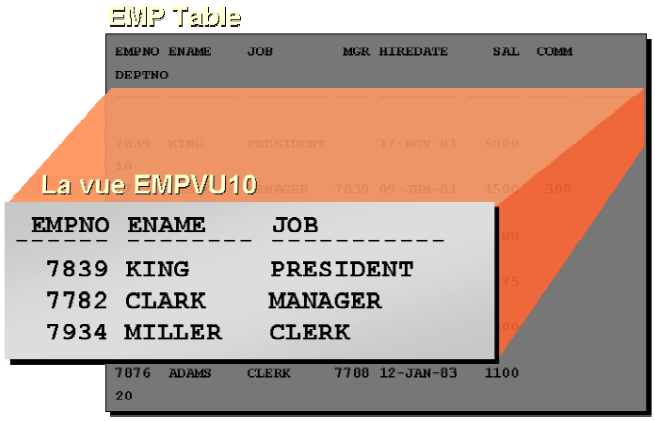
```
SQL> DROP SEQUENCE emp_empno;
```

5 Mise en place de vues

5.1 Administrer les vues

5.1.1 Introduction

Il est possible de représenter des combinaisons logiques ainsi que de grandes quantités de données en utilisant les vues.
Une vue est une structure logique basée sur une table ou une autre vue. Elle ne contient pas de données mais peut être assimilée à une "fenêtre" à travers laquelle les données d'une table (ou d'une autre vue) peuvent être sélectionnées ou modifiées.
Les tables sur lesquelles les vues se basent sont appelées des tables de base.



La création de vues permet de restreindre l'accès à la base évitant ainsi aux utilisateurs d'accéder directement aux données des tables. Les vues permettent de simplifier les requêtes des utilisateurs et de récupérer des données compliquées plus facilement sans pour autant écrire une jointure.

Les vues procurent donc une certaine indépendance vis à vis des utilisateurs non expérimentés.

Les vues peuvent aussi fournir un accès spécifique à chaque groupe d'utilisateurs en fonction de leurs critères particuliers, car on peut créer plusieurs vues des mêmes données. En effet plusieurs vues peuvent être créées sur les mêmes données.

Il y a deux types de vues:
- les vues simples
- les vues complexes

Option	***Vue Simple***	***Vue Complexe***
Nombre de table	Une	Une ou plusieurs
Contenir des fonctions	Non	Oui
Contenir un groupe de données	Non	Oui
Utilisation d'ordre DML à travers la vue	Oui	Pas toujours

La différence "basique" entre ces deux types est relative aux opérations DML.
Les vues simples dérivent de données d'une seule table et ne peuvent contenir de fonction de groupe, alors que les vues complexes sont basées sur plusieurs tables et peuvent contenir des fonctions de groupe.

5.1.2 L'ordre CREATE VIEW

Une vue est stockée dans le dictionnaire de données sous la forme d'un ordre SELECT.

Voici la syntaxe permettant de créer une vue:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nom_vue
[alias [,alias] …)]
AS requête_sql
[ WITH CHECK OPTION [CONSTRAINT contrainte]]
[ WITH READ ONLY];
```

REPLACE	Permet de remplacer directement la vue si celle ci existait déjà dans la base de données sans avoir à la supprimer.
FORCE	Permet de forcer la création de la vue même si la table, à laquelle elle fait référence n'existe pas dans la base de données.
alias	Permet de définir, lors de la création de la vue, des nouveaux noms pour les colonnes (leur nombre devra correspondre au nombre de colonnes retournées par la requête).
WITH CKECK OPTION	Permet d'autoriser les utilisateurs à insérer ou modifier seulement les lignes accessibles au travers de la vue.
WITH READ ONLY	Permet de supprimer le droit des utilisateurs à exécuter des opérations DML sur la vue.

Attention la requête de l'ordre SQL CREATE VIEW ne peut contenir la clause ORDER BY.

Il est possible de consulter la structure d'une vue grâce à la commande DESCRIBE.

Exemple:

```
SQL> CREATE OR REPLACE VIEW empvu_10
  2  AS
  3  SELECT *
  4  FROM emp
  5  WHERE deptno=10
  6  WITH READ ONLY;
```

5.1.3 Créer des vues simples

Exemple:

```
SQL> CREATE VIEW empvu10
  2  AS SELECT empno, ename, job
  3  FROM emp
  4  WHERE deptno=10;
```

Pour consulter les données d'une vue il est suffit d'utiliser la commande SELECT sur cette vue.

5.1.4 Créer des vues complexes

Exemple:

```
SQL> CREATE VIEW dept_sum_vu
  2  (name, minsal, maxsal, avgsal)
  3  AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)
  4  FROM emp e, dept d
  5  WHERE e.deptno=d.deptno
  6  GROUP BY d.dname;
```

5.1.5 Modifier une vue

Il est possible de modifier une vue en utilisation l'ordre CREATE OR REPLACE qui permet de remplacer le code d'une vue sans avoir à la supprimer.

Exemple :

```
SQL> CREATE OR REPLACE VIEW empvu10
  2  (employéé_number, employee_name, job_tittle)
  3  AS SELECT empno, ename, job
  4  FROM emp
  5  WHERE deptno=10;
```

5.1.6 Supprimer une vue

Suppression d'une vue :

```
DROP VIEW nom_vue;
```

Exemple:

```
SQL> DROP VIEW empvu_10;
```

5.2 Règles pour les opérations DML

5.2.1 Règles pour les opérations DML

Avec les ordres DML il est possible de supprimer, modifier ou rajouter des lignes dans une table. Ces opérations sont aussi possibles sur des vues simple et complexe mais avec des restrictions sur ces dernières.

Par exemple on ne pourra supprimer de ligne d'une vue complexe que si celle ci ne contient pas de fonction de groupe ou une clause GROUP BY ou bien utilise le mot DISTINCT dans sa requête de création.

On ne pourra pas modifier une vue si l'une de ses colonne est de type ROWNUM ou définie par une expression.

Enfin on ne pourra pas insérer de lignes dans les tables sous-jacentes à cette vue si la vue ne contient pas toutes les colonnes qui contiennent la contrainte NOT NULL et qu'il faudra obligatoirement remplir.

5.2.2 Règles pour les opérations DML sur des vues

Il existe certaines restrictions portant sur les ordres DML sur les vues complexes.
On ne pourra pas supprimer de lignes d'une vue complexe si cette vue contient des colonnes basées sur des fonctions de groupe ou bien GROUP BY ou bien DISTINCT.
On ne pourra pas modifier une vue si une colonne est une colonne ROWNUM ou bien si celle si est le résultat d'une expression (ex: SALARY*21).

5.2.3 La clause WITH CHECK OPTION

Nous allons maintenant voir comment restreindre les opérations DML possibles sur une vue. Pour cela il faudra rajouter la clause WITH CHECK OPTION dans l'ordre de création de la vue. Les commandes INSERT et UPDATE ne peuvent pas créer de lignes qui ne soient pas sélectionnable par la vue. Si l'utilisateur essayait de faire une opération DML sur une ligne qui ne puissent être affichée par la vue le serveur afficherait une erreur, l'erreur afficherai le nom de la contrainte ou sinon une erreur générique serait renvoyée.

5.2.4 Empêcher les opérations DML

Pour empêcher toute opérations DML sur une vue on rajoutera la clause WITH READ ONLY dans l'ordre de création de la table.

5.2.5 Afficher la définition des vues

Vous pourrez parfois avoir besoin de voir les détails d'une vue en particulier. Ces informations se trouvent dans la vue du dictionnaire USER_VIEWS et ALL_VIEWS.

6 Mise en place des synonymes et index

6.1 Création de synonymes

6.1.1 Créer un synonyme

Un synonyme est un nom alternatif pour designer un objet de la base de données. C'est aussi un objet de la base de données.
Pour désigner un objet appartenant à un autre schéma, il a été vu qu'il fallait le préfixer par le nom du schéma (ex: SCOTT.EMP). L'utilisation de synonyme permet de simplifier cette notation en supprimant le besoin de préfixer le nom de l'objet.

Voici la syntaxe de création d'un synonyme :

```
CREATE [PUBLIC] SYNONYM nom_synonyme
FOR object;
```

Par défaut un synonyme n'est accessible que du schéma dans lequel il a été crée.
La clause PUBLIC permet de définir si tout les utilisateurs pourront utiliser ce synonyme. Un synonyme public devra être unique dans la base.

Exemple:

```
SQL> CREATE PUBLIC SYNONYM d_num
  2  FOR dept_sum_table;
```

La commande CREATE OR REPLACE n'existe pas pour les synonymes.
Pour supprimer un synonyme on utilisera la commande :

```
DROP SYNONYM nom_synonyme;
```

6.2 Structure d'index

6.2.1 Introduction

Un index est un objet utilisé pour augmenter la recherche des éléments dans une table à l'aide d'un pointeur. Les index sont transparents pour l'utilisateur. Ils sont créés de manière automatique ou manuelle. Leur but principal est de réduire le nombre d'entrées/ sorties.
Quand une recherche sur une colonne, qui n'a pas été indexée, est effectuée, la requête fait une recherche sur chaque ligne et vérifie si les critères de recherche lui sont applicables si la colonne a été indexée, le serveur localise directement les lignes qui correspondent, d'où un gain de temps.
Il n'est pas nécessaire de nommer les index ni de les entretenir, c'est le serveur qui s'en occupera automatiquement.

Les indexes sont indépendant de manière logique et physique des tables qu'ils indexent. Il est donc possible de les créer et les supprimer sans aucunes répercutions sur les tables qu'ils indexent.

Il faut cependant être vigilant quand à l'utilisation des indexes car ils ne sont pas une garantie quand à l'augmentation de la vitesse. Car chaque opération DML effectuée force le serveur à mettre à jour les index. Donc, plus il y aura d'index, plus la mise à jour sera longue.

6.2.2 Types d'index

Il existe deux types d'index:

- Les index unique:
  Créés automatiquement lors de la définition d'une colonne comme étant une colonne PK ou UNIQUE. L'indexe prendra le nom de la contrainte à laquelle il est associé.

- Les index non unique:
  Créés manuellement. On pourra s'en servir pour indexer les colonnes FK afin d'accélérer la recherche des données. De plus ils pourront contenir des valeurs doubles (colonne non UNIQUE).

Ces deux index peuvent se baser sur une ou plusieurs colonnes. Un index multi-colonne pourra contenir jusqu'à 32 colonnes qui pourront ne pas être adjacentes (dans la même table).

6.3 Administrer les index

6.3.1 Créer un index

Prendre la décision de créer ou non un indexe est une décision importante. C'est généralement du ressort des DBAs. Il peut être intéressant de créer un index sur une colonne qui est utilisée fréquemment dans une clause WHERE ou lors d'une jointure.
Les colonnes contenant une grosse quantité de données ou qui contiennent de grosses quantités de valeurs nulle seront aussi de bonnes candidates.

Un indexe crée sur 2 ou plusieurs colonnes est appelé un indexe concaténé (concatenated index).

Il peut aussi être une bonne idée de créer un indexe sur de grosse table dont 5% des lignes sont utilisées.

Il n'est par contre pas intéressant de créer des indexes sur des petites tables (comme EMP ou DEPT), sur des colonnes peut utilisées ou sur des tables qui sont modifiées régulièrement ou sur des petites colonnes (<50 lignes) car cela diminue la vitesse des requêtes.

Voici la syntaxe pour créer un index :

```
CREATE INDEX nom_index
ON table(colonne [,colonne] …);
```

Exemple:

```
SQL> CREATE INDEX emp_deptno_idx
  2  ON emp (deptno);
```

6.3.2 Afficher les informations du dictionnaire de données

Les informations relatives aux indexes sont dans les vues du dictionnaire USER_INDEXES et USER_IND_COLUMNS.

De plus il existe différents types de vues du dictionnaire de données qui permettent de consulter ces informations sur différents niveaux. Ces vues sont préfixées de différentes manières et représentent un niveau de visibilité différent.

Préfixe	Description
USER_	Ces vues contiennent les informations relatives sur les objets appartenant à l'utilisateur en cours.
ALL_	Ces vues contiennent les informations relatives sur les objets appartenant à l'utilisateur mais aussi sur ceux qui lui sont

	accessible.
DBA_	Ces vues, seulement accessible par les DBA, contiennent toutes les informations sur tout les objets détenus par les utilisateurs.
V$_	Ces vues contiennent des informations sur les vues dynamiques de performances, sur les vues dynamiques du dictionnaire et sur les verrous.

6.3.3 Supprimer un index

On ne peut pas modifier un index pour modifier un indexe il faudra supprimer le premier puis en recréer un autre.
Voici la commande pour supprimer un index :

```
DROP INDEX nom;
```

Exemple:

```
SQL> DROP INDEX emp_deptno_idx;
```

7 Contrôle des accès des utilisateurs

7.1 Administration des privilèges système

7.1.1 Sécurité de la base de données

Gérer la sécurité de la base de données est le travail des DBA (Data Base Administrator). La sécurité est classée en deux catégories :

- La sécurité système:
  Permet de gérer les mécanismes qui contrôlent les accès et l'utilisation de la base au niveau système. Cette sécurité est implémentée grâce au nom d'utilisateur et à leur mot de passe. Ce type de sécurité permet aussi de définir des restrictions d'espace disque pour chaque utilisateur mais aussi de restreindre leur type d'action.
- La sécurité des données:
  Permet de gérer les mécanismes qui contrôlent les accès et l'utilisation de la base au niveau objet. Toutes ces sécurités sont accessible individuellement par les utilisateurs et permettent de définir les actions qu'ils pourront effectuer sur ces objets.

7.1.2 Créer un utilisateur

Créer des utilisateurs et leurs donner des pouvoirs permet aux DBAs de mieux gérer la sécurité. Lors de la création d'un utilisateur celui ci ne dispose d'aucun droit et ne peut donc même pas se connecter à la base de données.
Les droits et privilèges que vous allez lui donner vont définir les actions qu'il pourra effectuer. Voici la syntaxe qui va vous permettre de créer un utilisateur de base :

```
CREATE USER nom
IDENTIFIED BY password;
```

Exemple:

```
SQL> CREATE USER scott
  2  IDENTIFIED BY tiger;
```

7.1.3 Changer un mot de passe

Voici la commande pour changer de mot de passe :

```
ALTER USER        nom
IDENTIFIED BY     password;
```

Exemple:

```
SQL> ALTER USER scott
  2  IDENTIFIED BY toto;
```

7.1.4 Accorder un privilège

Voici quelques exemples de privilèges système (il en existe plus de 80 différents chacun possédant sa propre fonction).

Attention il vous faut parfois avoir plusieurs de ces privilèges pour pouvoir effectuer une opération de base. Les privilèges sont divisés en deux catégories :

- *Les privilèges utilisateur:*

CREATE SESSION	Permet de se connecter à la base de données. (privilège système)
CREATE TABLE	Permet de créer des tables dans son propre schéma.
CREATE SEQUENCE	Permet de créer ses propres séquences. (privilège système)
CREATE VIEW	Permet de créer ses propres vues.
CREATE PROCEDURE	Permet de créer ses propres procédures, fonctions et packages PL/SQL . (privilège système)

- *Les privilèges DBA : qui sont les plus puissant.*

CREATE USER	Création d'un utilisateur. (privilège système)
DROP USER	Suppression d'un utilisateur. (privilège système)
DROP ANY TABLE	Suppression de table dans tous les schémas. (privilège système)
BACKUP ANY TABLE	Sauvegarder n'importe quelle table. (privilège système)

Pour donner ces privilèges à un utilisateur vous allez utiliser la commande GRANT dont voici la syntaxe :

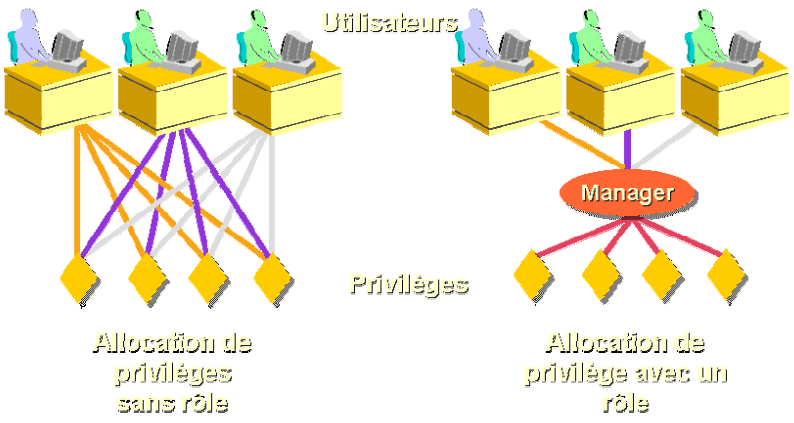
```
GRANT       privilège [,privilège2] …
TO          nom_utilisateur;
```

Exemple:

```
SQL> GRANT CONNECT,RESSOURCE
  2  TO scott;
```

7.1.5 Créer un rôle

Parfois il vous faudra donner un grand nombre de privilèges à des utilisateurs.
Cette tache longue et fastidieuse devra être répétée si vous devez leur rajouter un privilège supplémentaire.
Oracle vous offre une possibilité de réduire cette tache grâce à l'utilisation de rôles qui consistent en un ensemble de privilèges qui peuvent ensuite être donné à un utilisateur. De ce fait si vous avez donné ce rôle à tous les utilisateurs et que vous devez leur rajouter un privilège supplémentaire il vous suffira de le rajouter au rôle.



Voici la syntaxe pour créer un rôle :

```
CREATE ROLE nom_du_role;
```

Exemple:

```
SQL> CREATE ROLE TEST;
Rôle créé.

SQL> GRANT create session, create table, create view TO TEST;
Autorisation de privilèges (GRANT) acceptée.

SQL> GRANT TEST TO SCOTT;
Autorisation de privilèges (GRANT) acceptée.
```

Attention l'ordre CREATE OR REPLACE ROLE n'existe pas.
Vous venez dans cet exemple de donner les privilèges CREATE SESSION, CREATE TABLE, CREATE VIEW au rôle TEST.
Puis vous avez donné le rôle TEST à l'utilisateur SCOTT.

7.2 Administration les privilèges objets

7.2.1 Accorder des privilèges objet

Dès que vous créez un objet sur votre schéma vous héritez dès lors de tous les droits associés. Pour donner ces droits à d'autres utilisateurs vous devez utiliser la commande GRANT :

```
GRANT       privilège [[,privilege]|(columns)]
ON          object
TO          {user | role | PUBLIC};
```

Exemple:

```
SQL> GRANT SELECT
  2  ON emp(empno, ename, deptno)
  3  TO PUBLIC;
```

Si vous utiliser le mot clé PUBLIC, le privilège ou le rôle sera alors accordé à tous les utilisateurs de la base de données.

7.2.2 L'option WITH GRANT OPTION

Vous pouvez en plus de donner certains privilèges à un utilisateur vous pouvez donner à cet utilisateur le droit de donner les droits que vous lui avez donné à d'autres utilisateurs avec la commande WITH GRANT OPTION en fin de commande GRANT.
Si vous enlevez ensuite le privilège à cet utilisateur, tous les utilisateurs à qui il aura donné ce privilège se le verront enlevé aussi de manière automatique.

Il n'est pas possible d'utiliser WITH GRANT OPTION lors de l'attribution de privilèges à un rôle.

7.2.3 Vérifier les privilège accordés

Pour vérifier les droits que vous avez donnés en tant que DBA vous pouvez consulter les vues du dictionnaire suivantes :

ROLE_SYS_PRIVS	Contient les privilèges donnés à des rôles.
USER_ROLE_PRIVS	Contient les rôles accessibles par l'utilisateur.
USER_TAB_PRIVS_MADE	Contient tout les dons de droits sur des objets appartenant à l'utilisateur. (Tous les privilèges qu'il a donné sur ses objets)
USER_TAB_PRIVS_RECD	Contient toutes les informations sur les privilèges que l'utilisateur à

	reçu.
USER_COL_PRIVS_MADE	Contient tout les dons de droits sur des colonnes appartenant à l'utilisateur. (Tous les privilèges qu'il a donné sur des colonnes de ses objets)
USER_COL_PRIVS_RECD	Contient toutes les informations sur les privilèges que l'utilisateur a reçus sur des colonnes.

7.2.4 Révoquer un privilège

Pour supprimer un droit à un utilisateur ou à un rôle il vous suffira d'utiliser la commande suivante :

```
REVOKE        privilège [, privilège]
ON            nom_de_l'objet
FROM           user | role
[CASCADE CONSTRAINTS];
```

Exemple:

```
SQL> REVOKE RESSOURCE
  2  FROM SCOTT;
```

```
SQL> REVOKE SELECT
  2  ON emp
  3  FROM scott
  4  CASCADE CONSTRAINTS;
```