

# **Module n°3**

## **ORDRES DML & DDL**

*1Z0-007*

Auteur : Andrei LANGEAC  
Version 1.0 – 3 septembre 2004  
Nombre de pages : 52

# Sommaire

<b>1. MANIPULATION DES DONNEES (DML)</b> .....	<b>5</b>
1.1. INSERTION DES DONNEES .....	5
1.1.1. <i>La requête INSERT</i> .....	5
1.1.2. <i>Insertion des nouvelles lignes avec des valeurs nulles</i> .....	6
1.1.3. <i>Insertion de valeurs spéciales</i> .....	6
1.1.4. <i>Insertion des dates</i> .....	7
1.1.5. <i>La requête INSERT avec une sous-requête</i> .....	7
1.1.6. <i>Utilisation du mot clé DEFAULT</i> .....	8
1.1.7. <i>Création des scripts</i> .....	8
1.1.8. <i>Copie depuis une autre table</i> .....	8
1.2. MODIFICATION DES DONNEES .....	9
1.2.1. <i>La requête UPDATE</i> .....	9
1.2.2. <i>Mise à jour de deux colonnes à l'aide d'une sous-requête</i> .....	10
1.2.3. <i>Mise à jour des données d'une autre table</i> .....	10
1.2.4. <i>Utilisation du mot clé DEFAULT</i> .....	10
1.2.5. <i>Les contraintes d'intégrité lors de la mise à jour</i> .....	11
1.3. SUPPRESSION DES DONNEES .....	11
1.3.1. <i>La requête DELETE</i> .....	11
1.3.2. <i>Suppression des données d'une autre table</i> .....	11
1.3.3. <i>Les contraintes d'intégrité lors de la suppression</i> .....	12
1.4. UTILISATION DU MOT CLE WITH CHECK OPTION DANS UN ORDRE DML .....	12
1.5. LA REQUETE MERGE .....	13
1.6. PROCESSUS TRANSACTIONNEL .....	13
1.6.1. <i>L'ordre COMMIT</i> .....	13
1.6.2. <i>La requête ROLLBACK</i> .....	14
1.6.3. <i>Le processus transactionnel implicite</i> .....	15
1.6.4. <i>Verrouillage</i> .....	16
<b>2. CREATION ET MODIFICATION DES TABLES</b> .....	<b>17</b>
2.1. CREATIONS DES TABLES .....	17
2.1.1. <i>Nomenclature</i> .....	17
2.1.2. <i>La requête CREATE TABLE</i> .....	17
2.1.3. <i>L'option DEFAULT</i> .....	18
2.1.4. <i>Interrogation de dictionnaire des données</i> .....	18
2.1.5. <i>Différents types de données</i> .....	20
2.1.6. <i>Type de données date/heure</i> .....	21
2.1.7. <i>Création des tables en utilisant une sous-requête</i> .....	21
2.2. MODIFICATION DES TABLES .....	21
2.2.1. <i>La requête ALTER TABLE</i> .....	21
2.2.2. <i>Ajout d'une colonne</i> .....	22
2.2.3. <i>Modification de colonnes</i> .....	22
2.2.4. <i>Suppression de colonnes</i> .....	23
2.2.5. <i>L'option SET UNUSED</i> .....	23
2.3. SUPPRESSION DES TABLES .....	24
2.4. GESTION DES TABLES .....	24
2.4.1. <i>Changer le nom d'une table</i> .....	24
2.4.2. <i>Utilisation de la requête TRUNCATE sur une table</i> .....	24
2.4.3. <i>Ajout des commentaires sur une table</i> .....	25
<b>3. AJOUT DE CONTRAINTES</b> .....	<b>26</b>
3.1. QU'EST-CE QU'UNE CONTRAINTE .....	26
3.2. UTILISATION DES CONTRAINTES.....	26
3.2.1. <i>La contrainte NOT NULL</i> .....	26
3.2.2. <i>La contrainte UNIQUE</i> .....	27
3.2.3. <i>La contrainte PRIMARY KEY</i> .....	27

3.2.4.	<i>La contrainte FOREIGN KEY</i> .....	27
3.2.5.	<i>La contrainte CHECK</i> .....	27
3.3.	GESTION DES CONTRAINTES .....	28
3.3.1.	<i>Ajout d'une contrainte</i> .....	28
3.3.2.	<i>Suppression d'une contrainte</i> .....	28
3.3.3.	<i>Désactivation d'une contrainte</i> .....	28
3.3.4.	<i>Activation d'une contrainte</i> .....	29
3.3.5.	<i>Les contraintes en cascade</i> .....	29
3.3.6.	<i>Visualisation des contraintes</i> .....	29
<b>4.</b>	<b>CREATION DES VUES .....</b>	<b>31</b>
4.1.	PRESENTATION.....	31
4.1.1.	<i>Les objets de la base de données</i> .....	31
4.1.2.	<i>Qu'est-ce qu'une vue</i> .....	31
4.1.3.	<i>Pourquoi utiliser des vues?</i> .....	31
4.2.	GESTION DES VUES.....	32
4.2.1.	<i>Créer une vue</i> .....	32
4.2.2.	<i>Récupération des données depuis une vue</i> .....	33
4.2.3.	<i>Modification d'une vue</i> .....	34
4.2.4.	<i>Création d'une vue complexe</i> .....	35
4.2.5.	<i>Suppression d'une vue</i> .....	35
4.3.	OPERATIONS DML SUR UNE VUE .....	36
4.3.1.	<i>Règles d'exécution des opérations DML sur une vue</i> .....	36
4.3.2.	<i>Utilisation de la clause WITH CHECK OPTION</i> .....	36
4.3.3.	<i>Interdiction des opérations DML</i> .....	37
4.4.	LES VUES INLINE .....	38
4.5.	ANALYSE TOP-N.....	38
<b>5.</b>	<b>AUTRES OBJETS .....</b>	<b>40</b>
5.1.	SEQUENCES.....	40
5.1.1.	<i>Qu'est ce qu'une séquence</i> .....	40
5.1.2.	<i>Créer une séquence</i> .....	40
5.1.3.	<i>Vérification des séquences</i> .....	41
5.1.4.	<i>Les pseudo colonnes NEXTVAL et CURRVAL</i> .....	41
5.1.5.	<i>Utilisation des séquences</i> .....	42
5.1.6.	<i>Modifier une séquence</i> .....	42
5.1.7.	<i>Supprimer une séquence</i> .....	43
5.2.	INDEX.....	43
5.2.1.	<i>Qu'est ce qu'un index</i> .....	43
5.2.2.	<i>Création d'un index</i> .....	44
5.2.3.	<i>Vérification des index</i> .....	44
5.2.4.	<i>Index basé sur une fonction</i> .....	45
5.2.5.	<i>Supprimer un index</i> .....	45
5.3.	SYNONYMES .....	45
5.3.1.	<i>Qu'est ce qu'un synonyme</i> .....	45
5.3.2.	<i>Créer un synonyme</i> .....	45
5.3.3.	<i>Supprimer un synonyme</i> .....	46
<b>6.</b>	<b>CONTROLE D'ACCES DES UTILISATEURS .....</b>	<b>47</b>
6.1.	LES PRIVILEGES SYSTEME .....	47
6.1.1.	<i>Qu'est ce qu'un privilège</i> .....	47
6.1.2.	<i>Les privilèges DBA</i> .....	47
6.1.3.	<i>Créer un utilisateur</i> .....	47
6.1.4.	<i>Les privilèges système accordés à un utilisateur</i> .....	47
6.1.5.	<i>Accorder un privilège</i> .....	48
6.1.6.	<i>Créer et accorder un privilège à un rôle</i> .....	48
6.1.7.	<i>Modification de mot de passe</i> .....	49
6.2.	LES PRIVILEGES OBJET .....	49
6.2.1.	<i>Les privilèges objet</i> .....	49
6.2.2.	<i>Accorder les privilèges</i> .....	49

---

6.2.3. Utilisation des mots clés <i>WITH GRANT OPTION</i> et <i>PUBLIC</i> .....	50
6.2.4. Confirmation des privilèges accordés.....	51
6.2.5. Retirer les privilèges .....	51
6.3. LES LIENS SYMBOLIQUE DE BASE DE DONNEES.....	51

# 1. MANIPULATION DES DONNEES (DML)

## 1.1. Insertion des données


### 1.1.1. La requête INSERT

Cette requête permet d'insérer les données dans une table.

DEPARTMENTS			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

New row

...insert a new row into the DEPARTMENTS table...



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

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

<i>[(column_name [, column_name] ...)]</i>	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.
<i>VALUES (value [,value ...]);</i>	Permet de définir les valeurs à rajouter dans les colonnes de la table.

Lorsqu'on utilise un ordre **INSERT** avec la clause **VALUES**, une seule ligne est insérée dans la table. Si vous insérez une nouvelle ligne qui contient des valeurs pour chaque colonne, la liste de colonnes n'est pas obligatoire dans la clause **INSERT**. Cependant si la liste de colonnes n'est pas spécifiée, les valeurs doivent être entrées en respectant l'ordre par défaut des colonnes de la table.

Voici la structure de la table DEPT.

Nom	NULL ?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

Exemple:

```
SQL> INSERT INTO dept(deptno,dname,loc)
      2  VALUES(90,'Sciences','Paris');

1 ligne créée.
```

Explication: Cette requête **INSERT** une nouvelle ligne dans la table *DEPT*

### 1.1.2. Insertion des nouvelles lignes avec des valeurs nulles.

Il existe deux méthode pour insérer les valeurs nulles.

Méthode	Description
Implicite	Ne pas mettre la colonne dans la liste
Explicite	Spécifier le mot clé NULL dans la liste des valeurs, spécifier une chaîne vide (") dans la liste des valeurs pour les chaînes de caractères et dates.

Vous devez être sur que la colonne dans laquelle vous insérez les valeurs nulles ne possède pas la contrainte **NOT NULL**. Pour vérifier utiliser la commande **DESCRIBE** sur la table.

Méthode implicite:

```
SQL> INSERT INTO dept(deptno,dname)
      2  VALUES(5,'Implicite');

1 ligne créée.
```

Méthode explicite:

```
SQL>      INSERT INTO dept(deptno,dname,loc)
      2      VALUES(6,'Explicite',NULL);

1 ligne créée.
```

### 1.1.3. Insertion de valeurs spéciales

Vous pouvez utiliser des fonctions pour insérer les valeurs spéciales dans la table.

Exemple:

```
SQL>      INSERT INTO emp(empno,ename,hiredate)
      2      VALUES(1,'Robert',SYSDATE);

1 ligne créée.
```

Explication: Cette requête ajoute une nouvelle ligne à la table EMP. Grâce à **SYSDATE** la date du jour est insérée dans la colonne *hiredate*

Vous pouvez alors vérifier que l'ajout a été effectué.

```
SQL> SELECT ename,hiredate
2 FROM emp
3 WHERE empno=1;

      ENAME      HIREDATE
-----
Robert      10/08/04
```

#### 1.1.4. Insertion des dates

Si vous devez entrer la date dans un format autre que le format par défaut, (par exemple avec un siècle différent) vous devez utiliser la fonction **TO\_DATE**

Exemple:

```
SQL> INSERT INTO emp(empno,ename,hiredate)
2 VALUES(1,'MIKE',TO_DATE('FEV 3, 1999','MON DD, YYYY'));

1 ligne(s) créée(s).
```

Explication: Cette requête ajoute une nouvelle ligne à la table *EMP*. La valeur stockée dans la colonne *HIREDATE* est 3 février 1999.

Exemple:

```
SQL> INSERT INTO emp(empno,ename,hiredate)
2 VALUES(1,'MIKE','03-FEB-99');

1 ligne(s) créée(s).
```

Explication: Dans cette requête la valeur qui sera insérée dans la colonne *HIREDATE* sera 2099.

Lorsque vous utilisez le formatage *RR*, le système fournit automatiquement le bon siècle.

#### 1.1.5. La requête INSERT avec une sous-requête

Il est possible d'utiliser une sous-requête dans la clause **INSERT INTO** à la place du nom de la table. Le nombre de colonnes dans la clause **SELECT** de la sous-requête doit être identique au nombre de colonnes contenues dans la clause **VALUES**.

Exemple:

```
SQL> INSERT INTO
2 (SELECT empno,ename,hiredate,job,sal,deptno
3 FROM emp
4 WHERE deptno=30)
5 VALUES (3,'Taylor',TO_DATE('07-JUI-99','DD-MON-RR'),
        'ST_CLERK',5000,30);

1 ligne créée.
```

Vérification:

```
SQL>      SELECT      empno,ename,hiredate,sal , job,deptno
2         FROM        emp
3         WHERE       deptno=30;
```

EMPNO	ENAME	HIREDATE	SAL	JOB	DEPTNO
7499	ALLEN	20/02/81	1600	SALESMAN	30
7521	WARD	22/02/81	1250	SALESMAN	30
7654	MARTIN	28/09/81	1250	SALESMAN	30
7698	BLAKE	01/05/81	2850	MANAGER	30
7844	TURNER	08/09/81	1500	SALESMAN	30
3	Taylor	07/06/99	5000	ST_CLERK	30

### 1.1.6. Utilisation du mot clé DEFAULT

Grâce au mot clé **DEFAULT**, vous pouvez insérer la valeur par défaut définie sur la colonne. Si aucune valeur par défaut n'est définie, Oracle insère la valeur **NULL**.

Exemple :

```
SQL>      INSERT INTO      dept(deptno, dname)
3         VALUES          (56, DEFAULT) ;

1 ligne créée.
```

### 1.1.7. Création des scripts

Vous pouvez utiliser des fichiers scripts dans les requêtes **INSERT**. Pour ce faire, vous devez utiliser les variables de substitution.

Exemple:

```
SQL> INSERT INTO dept(deptno,dname,loc)
2   VALUES(&deptno, '&dname', '&loc');

Entrer une valeur pour deptno : 3

Entrer une valeur pour: Education

Entrer une valeur pour: New-York

old  2 : VALUES(&deptno, '&dname', '&loc')
new  2 : VALUES(3, 'Education', 'New-York')

1 ligne créée.
```

Explication: Cette requête ajoute des informations dans la table *DEPT* après avoir demandé les valeurs des variables de substitution aux utilisateurs.

### 1.1.8. Copie depuis une autre table

Vous pouvez ajouter des lignes dans une table lorsque ces lignes se situent dans une autre table. Vous devez utiliser une sous-requête à la place de la clause **VALUES**.



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

Le nombre de colonnes passées dans la clause **INTO** doit correspondre au nombre de colonnes sélectionnées dans la requête **SELECT**.

```
SQL> INSERT INTO emp(empno,sal,job)
  2  SELECT empno,sal,job
  3  FROM emp
  4  WHERE job='CLERK';

4 lignes créées.
```

## 1.2.Modification des données

### 1.2.1. La requête UPDATE

Il est possible grâce à l'ordre **UPDATE** de modifier des valeurs de colonnes dans les tables.

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

Exemple:

```
SQL> UPDATE emp
  2  SET sal=sal*1.20
  3  WHERE ename='SMITH';

1 ligne mise à jour..
```

Explication: Cette requête augmente le salaire de SMITH de 20%

Vous pouvez vérifier votre mise à jour en utilisant la requête **SELECT**

```
SQL> SELECT sal,ename
  2  FROM emp
  3  WHERE ename='SMITH';

      SAL  ENAME
-----
      960  SMITH
```

### 1.2.2. Mise à jour de deux colonnes à l'aide d'une sous-requête

Il est possible d'utiliser des sous-requêtes pour effectuer les mises à jour de plusieurs colonnes.

```
UPDATE table
  SET colonne =
      (SELECT colonne
       FROM table
       WHERE condition)

[colonne = (SELECT colonne
              FROM table
              WHERE condition)]
[WHERE condition ];
```

Exemple:

```
SQL> UPDATE emp
2 SET sal = sal * 1.20,
3 job = 'SALESMAN'
4 WHERE ename = 'SMITH';

1 ligne mise à jour.
```

Explication: Cette requête met à jour la table *EMP*, en augmentant le salaire de SMITH de 20% et en changeant son job à SALESMAN.

### 1.2.3. Mise à jour des données d'une autre table

Grâce à l'utilisation des sous-requêtes, on peut aussi mettre à jour les lignes d'une autre table.

Exemple:

```
SQL> UPDATE copy_emp
2 SET deptno = ( SELECT deptno
3 FROM emp
4 WHERE empno=7369 );

14 ligne mise à jour.
```

Explication: Cette requête met à jour la table *COPY\_EMP* basée sur les valeurs de la table *EMP*.

### 1.2.4. Utilisation du mot clé DEFAULT

Grâce au mot clé **DEFAULT**, vous pouvez utiliser la valeur par défaut définie sur la colonne. Si aucune valeur par défaut n'est définie, Oracle initialise la colonne à **NULL**.

Exemple :

```
SQL> UPDATE dept
2 SET loc = DEFAULT
3 WHERE deptno = 10;

1 ligne mise à jour.
```

### 1.2.5. Les contraintes d'intégrité lors de la mise à jour

Les contraintes d'intégrité permettent de vérifier que les données respectent un ensemble de règles définies. Donc si vous essayez de mettre à jour les valeurs qui possèdent des contraintes d'intégrité une erreur vous sera retournée.

Exemple :

```
SQL>      UPDATE      emp
  2         SET        deptno= 55
  3         WHERE     deptno = 10;
UPDATE emp
*
ERREUR à la ligne 1 :
ORA-02291: violation de contrainte (SCOTT.EMP_FK) d'intégrité -
touche parent introuvable
```

## 1.3. Suppression des données

### 1.3.1. La requête DELETE

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

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

Si vous utilisez **DELETE** sans la clause **WHERE**, toutes les lignes de la table seront supprimées.

Exemple:

```
SQL> DELETE FROM copy_emp;
14 ligne(s) supprimée(s).
```

Vous pouvez spécifier les lignes à supprimer dans la clause **WHERE**.

Exemple :

```
SQL> DELETE FROM copy_emp
  2  WHERE empno=7369;
1 ligne(s) supprimée(s).
```

### 1.3.2. Suppression des données d'une autre table

En utilisant une sous-requête dans la clause **WHERE**, vous pouvez supprimer les lignes de la table grâce aux valeurs basées sur une autre table.

Exemple:

```
SQL> DELETE FROM emp
2  WHERE deptno=(SELECT deptno
3                  FROM dept
4                  WHERE dname='RESEARCH');

5 ligne(s) supprimée(s).
```

Explication: Cette requête supprime les employés qui travaillent dans le département RESEARCH. La sous-requête recherche dans la table *DEPT*, le département nommé RESEARCH.

### 1.3.3. Les contraintes d'intégrité lors de la suppression

Vous ne pouvez pas supprimer une entrée qui possède une clé primaire référencée par une clé étrangère existante. La requête vous retournera une erreur, car on ne peut pas supprimer l'enregistrement parent qui possède des enregistrements enfant.

Exemple :

```
SQL> DELETE FROM dept
2  WHERE deptno = 10;
DELETE FROM dept
*
ERREUR à la ligne 1 :
ORA-02292: violation de contrainte (SCOTT.EMP_FK) d'intégrité -
enregistrement fils existant
```

Explication : On essaye de supprimer les lignes de la table *DEPT* lorsque *DEPTNO* est égal à 10. Cela n'est pas possible car la table *EMP* possède la colonne *DEPTNO*. Cette colonne est la clé étrangère de la colonne *DEPTNO* de la table *DEPT*.

## 1.4. Utilisation du mot clé WITH CHECK OPTION dans un ordre DML

Le mot clé **WITH CHECK OPTION** indique que si la sous-requête est utilisée à la place d'une table dans un ordre **INSERT**, **UPDATE**, **DELETE**. On ne pourra modifier que les lignes retournées par la sous-requête.

Exemple 1 :

```
SQL> INSERT INTO (SELECT empno,ename,hiredate,sal,deptno
2                  FROM emp
3                  WHERE deptno=30 WITH CHECK OPTION)
4  VALUES (9999, 'Smith', SYSDATE, 5000, 20);

FROM emp
*
ERREUR à la ligne 2 :
ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```

Exemple 2 :

```
SQL> INSERT INTO (SELECT empno,ename,hiredate,sal,deptno
2                FROM emp
3                WHERE deptno=30 WITH CHECK OPTION)
4  VALUES      (9999, 'Smith', SYSDATE, 5000, 30);

1 ligne(s) crée(s)
```

## 1.5. La requête MERGE

Grâce à cette requête, vous pouvez mettre à jour ou insérer des lignes conditionnellement dans la table. Ainsi cela permet plusieurs requêtes **UPDATE**. Si la ligne existe dans la table **MERGE** se comporte comme un **UPDATE**. Par contre si la ligne n'existe pas, **MERGE** se comporte comme un **INSERT**. Pour utiliser cette commande vous devez posséder les privilèges **INSERT** et **UPDATE** sur la table cible et le privilège **SELECT** sur la table source.

La requête **MERGE** est déterministe, vous ne pouvez pas mettre à jour la même ligne plusieurs fois dans la même requête **MERGE**.

```
MERGE INTO table AS table_alias
USING (table |vue/sous-requête) alias
ON (condition de jointure)
WHEN MATHED THEN
UPDATE SET
    Coll = coll_val1,
    Col2 = col2_val2
WHEN NOT MATCHED THEN
INSERT (column_list)
VALUES (column_values);
```

Exemple:

```
SQL> MERGE INTO copy_emp n
2     USING emp e
3     ON (n.empno = e.empno)
4     WHEN MATCHED THEN
5     UPDATE SET
6         n.ename = e.ename
7     WHEN NOT MATHED THEN
8     INSERT VALUES (e.empno, e.ename ,null, null, null, null,
null);

14 lignes fusionnées.
```

## 1.6. Processus transactionnel

### 1.6.1. L'ordre COMMIT

L'ordre **COMMIT** est utilisé pour terminer la transaction en cours en appliquant de façon permanente tous les changements.

Il est recommandé d'utiliser **COMMIT** après chaque ordre DML. Lorsque l'ordre **COMMIT** n'est pas utilisé, le serveur Oracle fait une sauvegarde de la table pour permettre aux autres utilisateurs de travailler dessus.

Exemple:

```
SQL> DELETE FROM emp;

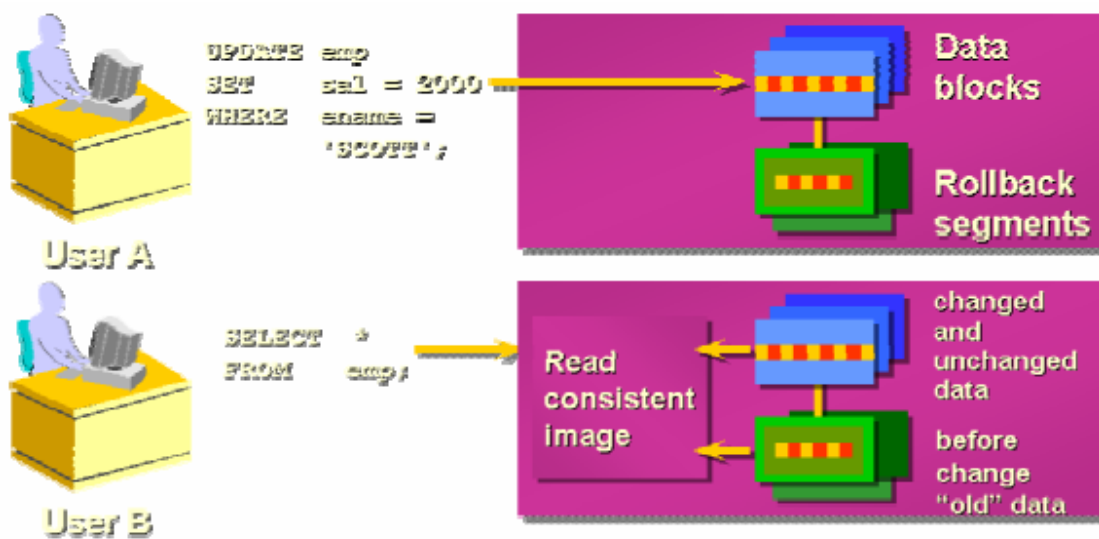
11 ligne(s) supprimée(s).

SQL> SELECT count(*) FROM emp;

  COUNT(*)
  -----
         0

SQL> COMMIT;

Validation effectuée
```



### 1.6.2. La requête ROLLBACK

**ROLLBACK** annule votre transaction et restitue l'état du serveur avant le changement.

```
SQL> DELETE FROM emp;

11 ligne(s) supprimée(s).

SQL> SELECT count(*) FROM emp;

  COUNT(*)
  -----
         0

SQL> ROLLBACK;

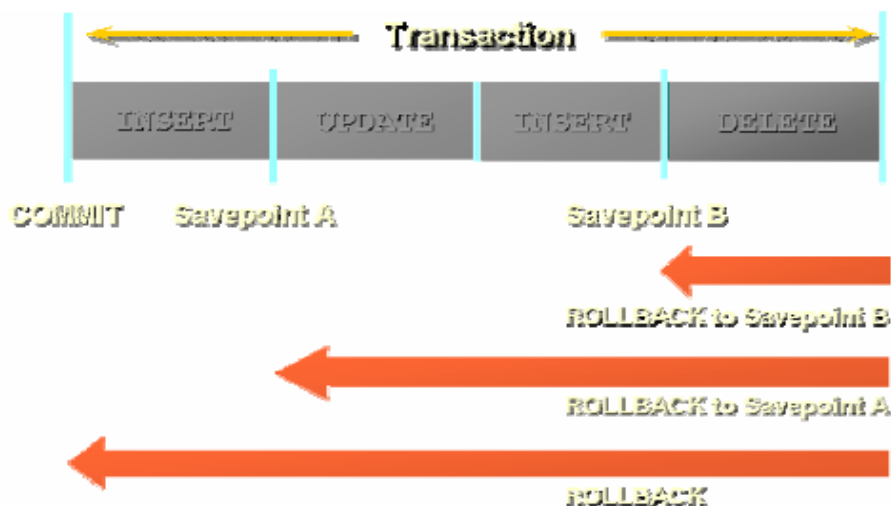
Annulation (rollback) effectuée.

SQL> SELECT COUNT(*) FROM emp;

  COUNT(*)
```

-----  
11

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 ;
```

### 1.6.3. Le processus transactionnel implicite

Statut	Circonstances
COMMIT automatique	Après les requêtes DDL ou DML. iSQL*PLUS se ferme normalement, sans le <b>COMMIT</b> ou <b>ROLLBACK</b> explicite
ROLLBACK automatique	Arrêt anormal d'iSQL*PLUS ou erreur du système.

Il existe une commande SQL\*Plus **AUTOCOMMIT** laquelle vous pouvez mettre en marche (ON ou OFF) qui valide les requêtes DML dès qu'elles s'exécutent.

### 1.6.4. Verrouillage.

Les mécanismes de verrouillage sont là pour prévenir l'interaction destructive entre différentes transactions qui accèdent à la même ressource.

Il existe deux modes de verrouillage :

- Share lock : ce verrouillage s'obtient automatiquement au niveau de la table lors des opérations DML. Plusieurs transactions peuvent avoir des share lock sur la même ressource.
- Exclusive lock : ce verrouillage s'obtient automatiquement pour chaque ligne de la table lors des opérations DML. Il empêche la modification de la ligne par une autre transaction jusqu'à ce que cette dernière soit **COMMIT** ou **ROLLBACK**.



## 2. CREATION ET MODIFICATION DES TABLES

### 2.1. Créations des tables

#### 2.1.1. Nomenclature

Il existe 6 règles pour nommer les tables et les colonnes en Oracle :

- L'identifiant doit commencer par une lettre
- L'identifiant 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, \_ , \$, #.**
- 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...
- Le nom des tables et des colonnes doit être explicite et représentatif des informations contenues dans la table et les colonnes.

#### 2.1.2. La requête 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 ], ... );
```

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

Exemple:

```
SQL> CREATE TABLE EMP2
2      (EMPNO NUMBER(2),
3      ENAME VARCHAR2(50),
4      LOCATION VARCHAR2(50) DEFAULT 'Not communicated');
Table créée.
```

Explication: On crée la table EMP2 avec trois colonnes EMPNO, ENAME et LOCATION

Pour vérifier que votre table a bien été créée utilisez la commande **DESCRIBE**.

```
SQL> DESC emp2;
Nom                                NULL ?   Type
-----
EMPNO                               NUMBER(2)
ENAME                               VARCHAR2(50)
LOCATION                              VARCHAR2(50)
```

### 2.1.3. L'option DEFAULT

Il est possible de donner une valeur par défaut à une colonne en utilisant la commande **DEFAULT** lors de la création de la table. Cette valeur peut être une valeur littérale, une expression ou une fonction SQL.

Exemple :

```
... hiredate DATE DEFAULT SYSDATE, ...
```

### 2.1.4. Interrogation de dictionnaire des données

Les dictionnaires de données sont divisés en 4 catégories

Préfixe	Description
<b>USER_</b>	Les vues qui contiennent des informations à propos des objets en possession des utilisateurs.
<b>ALL_</b>	Les vues qui contiennent des informations à propos des objets accessibles par un utilisateur
<b>DBA_</b>	L'accès à ces vues est réservé aux utilisateurs possédant le rôle de DBA.
<b>V\$</b>	Les vues de performance de la base de données.

Aperçu des tables dont l'utilisateur est propriétaire:

```
SQL> SELECT table_name
       2 FROM user_tables;

TABLE_NAME
-----
EMP2
EMP
DEPT
COPY_EMP
...
9 lignes sélectionnées.
```

Aperçu des différents objets dont l'utilisateur est propriétaire:

```
SQL> SELECT DISTINCT object_type
      2 FROM user_objects;

OBJECT_TYPE
-----
FUNCTION
INDEX
JAVA CLASS
JAVA SOURCE
PACKAGE
PACKAGE BODY
PROCEDURE
SEQUENCE
TABLE

9 lignes sélectionnées.
```

Aperçu des tables, vues, synonymes et séquences dont l'utilisateur est propriétaire :

```
SQL> SELECT *
      2 FROM user_catalog;

TABLE_NAME                                TABLE_TYPE
-----
EMP2                                       TABLE
ID_ARTICLE_SEQ                           SEQUENCE
ID_CARAC_SEQ                              SEQUENCE
ID_CAT_SEQ                                SEQUENCE
ID_CLIENT_SEQ                             SEQUENCE
ID_COMMANDE_SEQ                           SEQUENCE
ID_E_COMMANDE_SEQ                         SEQUENCE
T_ARTICLE                                 TABLE
T_CATEGORIE                              TABLE
T_CLIENT                                  TABLE
T_ENTETE_CARAC                            TABLE

TABLE_NAME                                TABLE_TYPE
-----
T_ENTETE_COMMANDE                        TABLE
T_INTER_COMMANDE                         TABLE
T_LANGAGE                                 TABLE
T_VALEUR_CARAC                            TABLE

15 lignes sélectionnées.
```

Aperçu de toutes les tables auxquels l'utilisateur a accès.

```
SQL> SELECT table_name
       2 FROM all_tables;

TABLE_NAME
-----
DUAL
SYSTEM_PRIVILEGE_MAP
TABLE_PRIVILEGE_MAP
STMT_AUDIT_OPTION_MAP
AUDIT_ACTIONS
PSTUBTBL
ODCI_SECOBJ$
ODCI_WARNINGS$
DEF$_TEMP$LOB
WM$WORKSPACES_TABLE
...
27 row(s) selected.
```

### 2.1.5. Différents types de données

Type de données	Description
<b>VARCHAR2 (taille)</b>	Texte de longueur variable. La longueur minimale est de 1 caractère et maximale 4000 caractères. La taille devra obligatoirement être définie.
<b>CHAR (taille)</b>	Texte de longueur fixes. La taille minimum de 1 caractère et pouvant aller jusqu'à 2000 caractères.
<b>NUMBER (p, e)</b>	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.)
<b>DATE</b>	Date notée sous la forme DD-MON-YY.
<b>LONG</b>	Texte de longueur variable pouvant stocker jusqu'à 2 gigas. On ne pourra mettre qu'une colonne de type LONG par table.
<b>RAW (taille)</b>	Equivalent à VARCHAR2, mais il permet de stocker des données binaires qui ne sont pas interprétées par Oracle. La taille maximum est de 2000.
<b>LONGRAW</b>	Equivalent à LONG mais pour des données de type binaire non interprétées par Oracle.
<b>CLOB</b>	Permet de stocker un pointeur vers un fichier de données composé de caractère et pouvant contenir jusqu'à 4 gigas.
<b>BLOB</b>	Permet de stocker un pointeur vers un fichier composé de données binaire et pouvant contenir jusqu'à 4 gigas.
<b>BFILE</b>	Permet de stocker les données binaires d'un fichier externe pouvant contenir jusqu'à 4 gigas.
<b>ROWID</b>	Le numéro système en base 64 qui représente l'adresse unique de la ligne dans la table

### 2.1.6. Type de données date/heure

Type de données DATE	Exemple	Description
TIMESTAMP	TIMESTAMP [(fractional_seconds_precision)]	Date avec des secondes fractionnées
TIMESTAMP WITH TIME ZONE	TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	Identique à TIMESTAMP, mais ajoute la différence entre l'heure locale et UTC
INTERVAL YEAR TO MONTH	INTERVAL YEAR [(year_precision)] TO MONTH	L'heure peut être stocké comme un intervalle d'années ou de mois
INTERVAL DAY TO SECOND	INTERVAL DAY [(day_precision)] TO SECOND	L'heure peut être stocké comme un intervalle de jours en heures, minutes ou secondes.

### 2.1.7. Création des tables en utilisant une sous-requête.

Il est possible de créer une table et insérer des lignes en utilisant la requête **CREATE TABLE** et l'option **AS sous-requête**.

```
CREATE TABLE   table
                  [(colonne, colonne ...)]
AS sous-requête ;
```

Exemple :

```
SQL> CREATE TABLE copy_emp AS SELECT * FROM emp ;

Table créée.
```

Explication : On crée une nouvelle table *copy\_emp* qui possède la même structure et les mêmes données que la table *emp*.

## 2.2. Modification des tables

### 2.2.1. La requête ALTER TABLE

Après avoir créé une table, il est parfois nécessaire de modifier sa structure. Pour modifier une table il faudra employer la commande : **ALTER TABLE** *nom\_de\_la\_table*

```
ALTER TABLE   tablename
ADD|MODIFY|DROP  (column datatype [DEFAULT expr]);
```

### 2.2.2. Ajout d'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);
```

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 présentes dans la table contenaient des valeurs, alors la nouvelle colonne sera initialisée à NULL pour toutes les lignes.

### 2.2.3. Modification de colonnes

Il existe 3 modifications de colonnes possibles :

- Modifier son type de données
- Modifier sa taille
- Modifier sa valeur par défaut

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

Règles de conduite :

- Vous pouvez augmenter la taille des colonnes
- Vous pouvez diminuer la taille des colonnes si les colonnes contiennent des valeurs nulles.
- Vous pouvez changer le type de dates si la colonne contient des valeurs nulles
- Vous pouvez convertir CHAR en VARCHAR2 ou VARCHAR2 en CHAR si la colonne contient des valeurs nulles ou si vous ne changez pas la taille.

Exemple:

```
SQL> ALTER TABLE dept2
2     MODIFY      (job VARCHAR(20));

Table modifiée.
```

Explication: Cette requête modifie la taille de la colonne JOB

### 2.2.4. Suppression de colonnes

Il est possible de supprimer des colonnes grâce à la requête **ALTER TABLE** et à la clause **DROP COLUMN**

Voici 2 syntaxes que vous pouvez utiliser:

```
ALTER TABLE nom_table  
DROP COLUMN nom_colonne
```

```
ALTER TABLE nom_table  
DROP (nom_colonne)
```

Exemple :

```
SQL> ALTER TABLE employee  
2 DROP COLUMN birth ;
```

Il n'est possible de supprimer 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.

### 2.2.5. L'option SET UNUSED

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 UNUSED last_name;
```

Cet ordre 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.

## 2.3. Suppression des tables

La commande **DROP TABLE** supprime définitivement la définition de la table. Lorsque vous supprimez une table:

- Toutes les données sont supprimées.
- Toutes les transactions sont committées.
- Tous les indexes sont supprimés.
- Vous ne pouvez pas effectuer ROLLBACK.

**DROP TABLE** *nom\_table* ;

Exemple :

```
SQL> DROP TABLE emp;
Table supprimée.
```

## 2.4. Gestion des tables

### 2.4.1. Changer le nom d'une table

Grâce à la commande **RENAME**, vous pouvez changer les noms des tables, des vues, des séquences, ou des synonymes. Pour l'utiliser vous devez être le propriétaire de l'objet que vous voulez renommer

**RENAME** *ancien\_nom\_table*  
**TO** *nouveau\_nom\_table*;

Exemple:

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

### 2.4.2. Utilisation de la requête TRUNCATE sur une table

La commande **TRUNCATE** permet de vider une table de toutes ses données tout en conservant sa structure :

**TRUNCATE TABLE** *nom\_table*;

Vous pouvez aussi supprimer toutes les lignes grâce à la commande **DELETE**.

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

Exemple:

```
SQL> TRUNCATE employee;
```

```
SQL> DELETE employee;
```



### 2.4.3. Ajout des commentaires sur 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.

- **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 ' ').

## 3. AJOUT DE CONTRAINTES

### 3.1. Qu'est-ce qu'une contrainte

Oracle utilise des contraintes pour empêcher l'utilisateur d'entrer des données invalides dans la base.

Règles de conduite :

- Toutes les contraintes sont stockées dans le dictionnaire de données.
- Donner des noms explicites à vos contraintes pour faciliter le référencement.
- Les contraintes peuvent être définies soit au moment de la création de la table soit après la création.

Voici les différentes contraintes :

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. CHECK

```
CREATE TABLE table
  (colonne datatype[DEFAULT expr]
  [colonne_contrainte],
  ...
  [table_contrainte][,...]);
```

### 3.2. Utilisation des contraintes

#### 3.2.1. La contrainte NOT NULL

Cette contrainte spécifie que la colonne ne peut pas contenir des valeurs nulles. La contrainte **NOT NULL** peut être spécifiée uniquement sur la colonne.

Exemple :

```
CREATE TABLE employee (
  employee_id NUMBER(6),
  nom VARCHAR2(25) NOT NULL, ←NOMME PAR LE SYSTEME
  sal NUMBER(8,2),
  hiredate DATE CONSTRAINT emp_hiredate_nn NOT NULL, ←
  NOMME PAR L'UTILISATEUR
  ...);
```

### 3.2.2. La contrainte UNIQUE

Cette contrainte spécifie la ou les colonne(s) dont les valeurs sont uniques pour toutes les lignes de la table.

```
SQL> CREATE TABLE department2(
2          email VARCHAR2(30) CONSTRAINT dept2_email_u UNIQUE);
Table créée.
```

### 3.2.3. La contrainte PRIMARY KEY

La contrainte **PRIMARY KEY** permet de créer la clé primaire de la table. On ne peut créer une seule clé par table.

```
SQL> CREATE TABLE emp(
2          empno NUMBER(8) PRIMARY KEY,
3          ename VARCHAR2(50),
4          email VARCHAR2(50) UNIQUE);
Table créée.
```

### 3.2.4. La contrainte FOREIGN KEY

Cette contrainte établit une relation de clé étrangère entre une colonne et la colonne dans la table référencée.

Mot clé	Description
<b>FOREIGN KEY</b>	Définit la colonne sur laquelle portera la clé étrangère
<b>REFERENCES</b>	Identifie la table et la colonne dans la table parent
<b>ON DELETE CASCADE</b>	Supprime les lignes de la table enfant lorsque les lignes de la table parent sont supprimés
<b>ON DELETE SET NULL</b>	Convertie les clés étrangères dépendantes à NULL.

### 3.2.5. La contrainte CHECK

Cette contrainte définit la condition que doit satisfaire chaque ligne de la table.

Les expressions suivantes ne sont pas autorisées :

- Les pseudo colonnes **CURRVAL**, **NEXTVAL**, **LEVEL**, **ROWNUM**
- L'appel à des fonctions **SYSDATE**, **UID**, **USER** et **USERENV**
- Les requêtes se referant aux autres valeurs dans les autres tables.

Exemple :

```
CREATE TABLE employee (
...
sal NUMBER(8,2) CONSTRAINT emp_sal_min CHECK (sal >0),
...);
```

## 3.3. Gestion des contraintes

### 3.3.1. Ajout d'une contrainte

Vous pouvez ajouter une contrainte en utilisant **ALTER TABLE** et **ADD CONSTRAINT**

Voici la syntaxe :

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

Exemple :

```
SQL> ALTER TABLE emp  
2     ADD CONSTRAINT emp_manager_FK  
3     FOREIGN KEY(manager_id)  
4     REFERENCES emp(empno) ;
```

### 3.3.2. Suppression d'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 ;
```

Si vous voulez supprimer la contrainte **PRIMARY KEY** utilisez le mot clé **CASCADE** pour supprimer les **FOREIGN KEY** qui s'y réfèrent.

### 3.3.3. Désactivation d'une contrainte

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 CONSTRAINT nom_contrainte  
[CASCADE]
```

Exemple:

```
SQL> ALTER TABLE emp3
2 DISABLE CONSTRAINT emp3_pk;

Table modifiée.
```

### 3.3.4. Activation d'une contrainte

Après avoir désactivé une contrainte, il est possible de la réactiver

```
ALTER TABLE nom_table
ENABLE CONSTRAINT nom_contrainte
[CASCADE]
```

Voici les règles à respecter pour activer une contrainte :

1. Toutes les données de la table doivent respecter la contrainte.
2. Lorsque vous activez la contrainte **UNIQUE** ou **PRIMARY KEY**, l'index **UNIQUE** et **PRIMARY KEY** est créé automatiquement.
3. Lorsque vous activez la contrainte **PRIMARY KEY** avec l'option **CASCADE** la contrainte **FOREIGN KEY** n'est pas activée.

Exemple:

```
SQL> ALTER TABLE emp3
2 ENABLE CONSTRAINT emp3_pk;
```

### 3.3.5. Les contraintes en cascade

La clause **CASCADE CONSTRAINTS** est utilisée avec la clause **DROP COLUMN**. Cette clause permet de supprimer les contraintes se référant à des clés primaires dans les colonnes supprimées. Il est aussi possible de supprimer les contraintes multi-colonnes définies sur les colonnes supprimées.

Exemple :

```
SQL> ALTER TABLE dept
2 DROP (deptno) CASCADE CONSTRAINTS ;
```

### 3.3.6. Visualisation des contraintes

Avec la commande **DESCRIBE**, vous pouvez visualiser l'existence de la contrainte **NOT NULL**. Pour voir les autres contraintes, vous devez faire une requête sur la table **USER\_CONSTRAINTS**

Exemple:

```
SQL> SELECT constraint_name, constraint_type
2 FROM user_constraints
3 WHERE table_name='DEPT';

CONSTRAINT_NAME          C
-----
DEPT_DEPTNO_PK           P
```

Explication: Cette requête affiche les contraintes de la table *DEPT*.

Si l'utilisateur ne spécifie pas le nom de la contrainte, le serveur Oracle le fera automatiquement. Vous pouvez voir les noms des colonnes qui possèdent des contraintes, en faisant une requête sur la vue `USER_CONS_COLUMNS`

```
SQL> SELECT      constraint_name, column_name
2  FROM          user_cons_columns
3  WHERE         table_name= 'DEPT' ;
```

CONSTRAINT_NAME	COLUMN_NAME
DEPT_DEPTNO_PK	DEPTNO

## 4. CREATION DES VUES

### 4.1. Présentation

#### 4.1.1. Les objets de la base de données

Objet	Description
Table	L'unité de stockage de base composée de lignes et de colonnes.
Vue	Représentation logique des données d'une ou de plusieurs tables.
Séquence	Génère des valeurs des clés primaires
Index	Améliore la performance de certaines requêtes
Synonyme	Un nom alternatif pour certains objets.

#### 4.1.2. Qu'est-ce qu'une vue

##### EMPLOYEES Table:

EMPLOYEE ID	FIRST NAME	LAST NAME	EMAIL	PHONE NUMBER	HIRE DATE	JOB ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
							8000
							4300
							5900
							3500
							3100
							2600
							2500
							10500
							11000
							8800
							13000
							6000
							12000
							6300
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	6300

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

La vue est basée sur une table ou une autre vue. Elle ne contient pas de données.

Une vue est une sorte de fenêtre par laquelle des données peuvent être sélectionnées et changées.

#### 4.1.3. Pourquoi utiliser des vues?

##### Les avantages des vues

- Les vues restreignent l'accès aux données, car les vues peuvent afficher certaines colonnes de la table.
- Les vues peuvent être utilisées pour faire les requêtes simples pour retrouver des résultats des requêtes compliquées.

- Une vue peut être utilisée pour afficher des données de plusieurs tables.

Voici deux classifications pour les vues. Des vues simples et complexes.

Élément	Simple	Complexe
Nombre de tables	Une	Une ou plusieurs
Contient des fonctions	Non	OUI
Contient des groupes de données	Non	Oui
Opérations DML	Oui	Pas toujours

## 4.2. Gestion des vues

### 4.2.1. Créer une vue

Pour créer une vue vous devez utiliser **CREATE VIEW**

```

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

- OR REPLACE:** Recrée le vue si elle existe.
- FORCE:** Crée une vue même si la table n'existe pas.
- NOFORCE:** Crée une vue même si la table existe.
- Viewname:** Nom de la vue.
- Alias:** Spécifie des noms pour les expressions sélectionnées
- Subquery:** La requête SELECT complète.
- WITH CHECK OPTION:** Spécifie que seul des lignes accessibles par la vue peuvent être mis à jour ou insérés.
- Constraint:** Est le nom donné à la contrainte **CHECK OPTIONS**.
- WITH READ ONLY:** Assure qu'aucune opération DML ne peut pas être effectuée.

Exemple:

```

SQL> CREATE OR REPLACE VIEW empvu
2 AS
3 SELECT empno, ename, sal
4 FROM emp
5 WHERE deptno=30;

Vue créée.
  
```



Explication: Cette requête crée la vue EMPVU basée sur la table EMP

Grâce à la commande DESCRIBE vous pouvez vous assurer que la vue a été créée.

```
SQL> DESCRIBE empvu;

Name                                NULL ?   Type
-----                                -
NUMBERID                             NOT NULL NUMBER(4)
LASTNAME                              VARCHA2(10)
SALARY                                 NUMBER(7,2)
```

Il est possible de contrôler les noms des colonnes en incluant des alias de colonnes dans la sous requête.

Le nombre d'alias doit correspondre au nombre de colonnes sélectionné par la sous-requête.

Exemple:

```
SQL> CREATE OR REPLACE VIEW empvu
2 AS
3 SELECT empno number_id,ename last_name, sal
4 FROM emp
5 WHERE deptno=30;

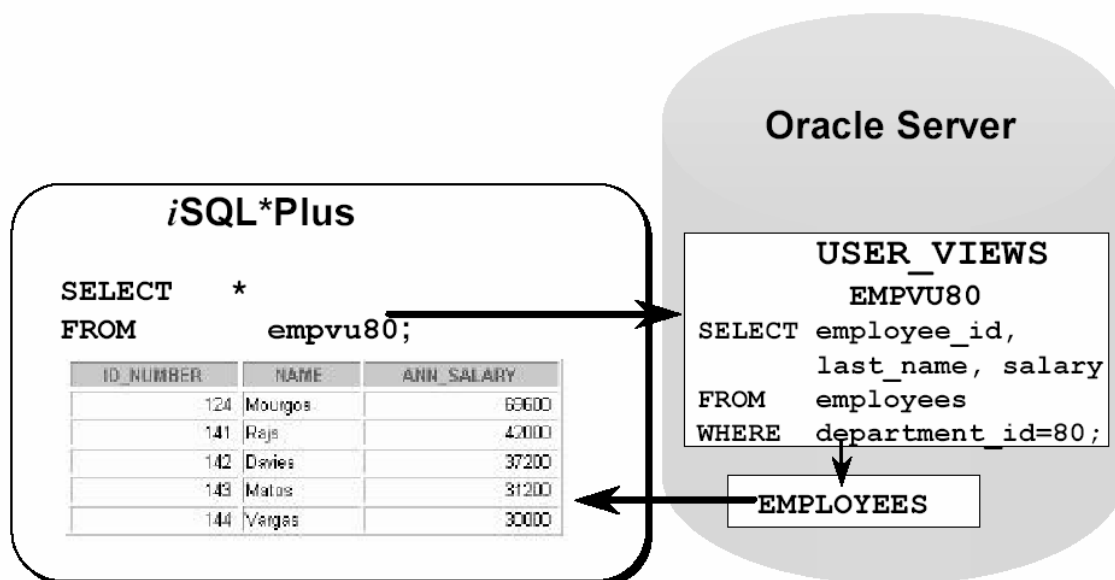
View created.
```

Explication: Cette requête crée la vue *EMPVU2* basée sur la table *EMP*. Elle contient le numéro de l'employé, le nom, le salaire pour les personnes travaillant dans le département 30, avec les alias *NUMBER\_ID* et *LAST\_NAME*.

#### 4.2.2. Récupération des données depuis une vue

La récupération des données depuis une vue se fait grâce à un simple ordre **SELECT**

Lorsqu'un utilisateur accède à des données à partir d'une vue, le serveur Oracle retrouve la définition de la vue dans la table *USER\_VIEWS*. Ensuite il vérifie les privilèges pour la vue et enfin exécute l'opération sur la table.



### 4.2.3. Modification d'une vue

Si vous avez créé une vue avec l'option **OR REPLACE**, il est possible de modifier la structure de la vue en la recréant sans avoir à la supprimer.

Exemple:

```

SQL> DESCRIBE empvu;

Name                               NULL ?   Type
-----
EMPNO                               NOT NULL NUMBER(4)
ENAME                               VARCHAR2(10)
SAL                                  NUMBER(7,2)

```

Explications: La vue *EMPVU* est créée sans les alias de colonnes.

Exemple:

```

SQL> CREATE OR REPLACE VIEW empvu
2 AS
3 SELECT empno, ename, sal, comm
4 FROM emp
5 WHERE deptno=30;

Vue créée.

```

Explication: Cette requête modifie la vue *EMPVU* en ajoutant la colonne **COMMISSION**

Vous pouvez vérifier que les changements ont été effectués grâce à la commande **DESCRIBE**.

```
SQL> DESC empvu
Nom                                NULL ?   Type
-----
EMPNO                               NOT NULL NUMBER(4)
ENAME                               VARCHAR2(10)
SAL                                  NUMBER(7,2)
COMM                                 NUMBER(7,2)
```

#### 4.2.4. Création d'une vue complexe

Il existe deux types de vues: simples et complexes.

Exemple:

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

Vue créée.
```

Explication: La vue comporte les noms de départements, le salaire minimum, le salaire maximum et la moyenne de salaire par département.

Vous pouvez vérifier que la vue complexe a été créée grâce à la commande DESCRIBE.

```
SQL> DESCRIBE deptvu;

Nom                                NULL ?   Type
-----
NAME                               VARCHAR2(14)
MINSAL                             NUMBER
MAXSAL                             NUMBER
AVGSAL                             NUMBER
```

#### 4.2.5. Suppression d'une vue

Lorsque vous supprimez une vue, les données ne sont pas perdues.

Pour supprimer une vue utilisez **DROP VIEW**

```
DROP VIEW      viewname;
```

Vous ne pouvez pas rollback cette opération.

Exemple:

```
SQL> DROP VIEW deptvu;

Vue supprimée.
```

## 4.3. Opérations DML sur une vue

### 4.3.1. Règles d'exécution des opérations DML sur une vue

Vous pouvez exécuter des opérations DML sur des vues simples, mais il n'est pas possible de supprimer des lignes qui contiennent:

- Des fonctions de groupe.
- La clause **GROUP BY**
- Le mot clé **DISTINCT**.
- Le mot clé de la pseudo colonne **ROWNUM**.

Vous pouvez ajouter des données grâce à une vue si elle ne contient pas des objets listés précédemment ou s'il n'y a pas de colonnes **NOT NULL** sans valeurs par défaut qui ne sont pas sélectionné par la vue.

Noter que vous ajoutez des données directement dans la table.

### 4.3.2. Utilisation de la clause WITH CHECK OPTION

Grâce à clause **WITH CHECK OPTION**, vous spécifiez que les requêtes **INSERT** et **UPDATE** ne peuvent pas créer des enregistrements ne pouvant être sélectionnés par la vue.

Exemple:

```
SQL> CREATE OR REPLACE VIEW empvu2
2 AS
3 SELECT *
4 FROM emp
5 WHERE deptno=20
6 WITH CHECK OPTION CONSTRAINT empvu20_ck;

Vue créée.
```

Explication: La vue *EMPVU2* est créé avec l'option **WITH CHECK OPTION**

Exemple:

```
SQL> UPDATE empvu2
2 SET deptno=10;

UPDATE empvu2
*
ERREUR à la ligne 1 :
ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```

Explication: La requête ne s'exécute pas car on essaye de changer l'id de département à 10.

### 4.3.3. Interdiction des opérations DML

Avec l'option **WITH READ ONLY**, vous pouvez vous assurer qu'aucune opération DML ne sera exécutée sur la vue.

Le serveur Oracle vous retournera une erreur lorsqu'une opération DML sera exécutée.

Exemple:

```
SQL> CREATE OR REPLACE VIEW deptvu3
 2 AS
 3 SELECT *
 4 FROM dept
 5 WITH READ ONLY CONSTRAINT deptvu3_ck;

Vue créée.
```

Exemples:

```
SQL> UPDATE deptvu3
 2 SET dname='Oracle';

SET dname='Oracle'
*
ERREUR à la ligne 2 :
ORA-01733: les colonnes virtuelles ne sont pas autorisées ici
```

```
SQL> DELETE FROM deptvu3
 2 WHERE deptno=10;

DELETE FROM deptvu3
*
ERREUR à la ligne 1 :
ORA-01752: Impossible de supprimer de la vue sans exactement une
table protégée par clé
```

Explication: Une erreur est retournée car on essaye d'exécuter un ordre DML

## 4.4. Les vues inline

Une vue **INLINE** est une sous requête avec un alias.

Ce type de vue est créé lorsque vous placez une sous-requête dans la clause **FROM**

Exemple:

```
SQL> SELECT      a.ename, a.sal, a.deptno, b.maxsal
 2 FROM          emp a,
                (SELECT deptno,MAX(sal) maxsal
 3                  FROM emp
 4                  GROUP BY deptno) b

 5 WHERE         a.deptno=b.deptno
 6 AND           a.sal <b.maxsal ;
```

ENAME	SAL	DEPTNO	MAXSAL
CLARK	2450	10	5000
MILLER	1300	10	5000
SMITH	800	20	3000
ADAMS	1100	20	3000
JONES	2975	20	3000
ALLEN	1600	30	2850
MARTIN	1250	30	2850
JAMES	950	30	2850
TURNER	1500	30	2850
WARD	1250	30	2850

10 lignes sélectionnées.

Explication: La vue *b* retourne les numéros de départements et la salaire maximum pour chaque département de la table EMP.

Grâce à la clause **WHERE a.deptno=d.deptno AND a.sal <b.maxsal**, on affiche les noms des employés, le salaire, les numéros de départements, et le salaire maximum pour tous les employés qui gagnent moins que la personne qui possède le salaire maximum dans leur département.

## 4.5. Analyse TOP-N

Vous pouvez choisir d'afficher uniquement les *N* premiers ou les *N* derniers enregistrements de la table grâce à la requête **TOP-N**

Par exemple vous pouvez afficher:

- Les trois premiers salariés de l'entreprise.
- Les quatre employés embauchés récemment.
- Les trois produits les plus vendus.

```

SELECT      [column list], ROWNUM
FROM

                (SELECT      [column list]
                 FROM          table
                 ORDER BY Top-n column)

```

**WHERE ROWNUM** *condition*;

Subquery:                   Ou la vue inline qui sélectionne les données.

Requête principale:       Limite le nombre de lignes du résultat final.

ROWNUM:                    Pseudo colonne qui assigne la valeur séquentielle.

WHERE:                     Spécifie les n lignes qui seront retournés.

Exemple:

```

SQL> SELECT      ROWNUM, e.ename,e.hiredate
2 FROM          (SELECT ename, hiredate
3              FROM emp
4              ORDER BY hiredate) e
5 WHERE ROWNUM < = 4;

      ROWNUM ENAME      HIREDATE
-----
          1 SMITH      17/12/80
          2 ALLEN      20/02/81
          3 WARD       22/02/81
          4 JONES      02/04/81

```

Explication: Cette requête affiche les noms et les salaires des top trois employées de la table *EMP*.

La sous-requête retourne les noms et les salaires de la table *EMP*.

**WHERE ROWNUM <=4** permet d'afficher les 3 premiers résultats.

## 5. AUTRES OBJETS

### 5.1. Séquences

#### 5.1.1. Qu'est ce qu'une séquence

Une séquence est un objet créé par l'utilisateur. Elle sert à créer des valeurs pour les clés primaires, qui sont incrémentées ou décrémentées par le serveur Oracle.

Noter que la séquence est stockée et générée indépendamment de la table, et une séquence peut être utilisée pour plusieurs tables.

#### 5.1.2. Créer une séquence

```
CREATE SEQUENCE nomsequence  
  [INCREMENT BY n]  
  [START WITH n]  
  [{MAXVALUE n}]  
  [{MINVALUE n}]  
  [{CYCLE | NOCYCLE}]  
  [{CACHE n | NOCACHE}];
```

**INCREMENT BY n:** Spécifie l'intervalle d'incrément (Si n n'est pas spécifié n=1)

**START WITH n:** Spécifie la valeur de départ. (Si n n'est pas spécifié n=)

**MAXVALUE n:** Spécifie la valeur maximale.

**MINVALUE n:** Spécifie la valeur minimale.

**CYCLE | NOCYCLE:** Lorsque la séquence a atteint sa valeurs maximale elle est régénéré. (NOCYCLE est la valeur par défaut)

**CACHE n | NOCACHE:** Spécifie le nombre de valeurs que le serveur Oracle garde en mémoire.

Exemple :

```
SQL> CREATE SEQUENCE dept_deptno_seq  
  2   INCREMENT BY 10  
  3   START WITH 10  
  4   NOCYCLE  
  5   NOCACHE;  
  
Séquence créée.
```

Explication : Cette requête crée la séquence DEPT\_DEPTNO\_SEQ qui sera utilisée sur la colonne *deptno* de la table *DEPT*. La séquence commence à 10, s'incrémente par 10 avec les options NOCYCLE et NOCACHE.



### 5.1.3. Vérification des séquences

Pour vérifier que la séquence a été créée, interrogez le dictionnaire de données USER\_OBJECTS.

```
SQL> SELECT      object_name
2 FROM          user_objects
3 WHERE         object_type='SEQUENCE' ;

OBJECT_NAME
-----

DEPT_DEPTNO_SEQ
EDITEUR_ID_SEQ
ID_CIRCUIT_SEQ
ID_CLIENT_SEQ
ID_ETAPES_SEQ
ID_HOTEL_SEQ
ID_SITE_TOURISTIQUE_SEQ
...
```

Vous pouvez aussi retrouver des informations sur la séquence dans USER\_SEQUENCES.

```
SQL> SELECT      min_value, max_value, increment_by, last_number
2 FROM          user_sequences
3 WHERE         sequence_name='DEPT_DEPTNO_SEQ' ;

MIN_VALUE  MAX_VALUE  INCREMENT_BY  LAST_NUMBER
-----
1 1,0000E+27          10          10
```

### 5.1.4. Les pseudo colonnes NEXTVAL et CURRVAL

Après avoir créée une séquence, vous pouvez utiliser les pseudo colonnes **NEXTVAL** et **CURRVAL** pour rechercher la valeur suivante ou la valeur courante de la séquence.

Exemple :

```
SQL> SELECT      dept_deptno_seq.NEXTVAL
2 FROM          dual;

NEXTVAL
-----
30
```

Exemple :

```
SQL> SELECT      dept_deptno_seq.NEXTVAL
2 FROM          dual;

NEXTVAL
-----
50

SQL> SELECT      dept_deptno_seq.CURRVAL
2 FROM          dual;

CURRVAL
-----
50
```

Si la séquence est appelée pour la première fois, vous devez utiliser la pseudo colonne **NEXTVAL** avant l'utilisation de **CURRVAL**.

**Vous pouvez utiliser NEXTVAL et CURRVAL dans les cas suivants:**

- La requête **SELECT** ne fait pas partie de la sous-requête.
- Dans la requête **SELECT** faisant partie de la sous-requête dans un ordre **INSERT**.
- Dans la clause **VALUES** de la requête **INSERT**.
- Dans la clause **SET** de la requête **UPDATE**.

**Vous ne pouvez pas utiliser NEXTVAL et CURRVAL dans les cas suivants:**

- Lors d'ordre **SELECT** sur une vue.
- La requête **SELECT** possédant le mot clé **DISTINCT**.
- Ordre **SELECT** possédant les clauses **GROUP BY**, **ORDER**, ou **HAVING**
- Dans une sous-requête appartenant aux ordres **SELECT**, **UPDATE** ou **DELETE**.
- Dans une expression **DEFAULT** lorsqu'on utilise **CREATE TABLE** ou **ALTER TABLE**.

### 5.1.5. Utilisation des séquences

Les séquences peuvent être utilisées pour insérer des valeurs dans la table lors d'une requête **INSERT INTO**.

Exemple :

```
SQL> INSERT INTO dept
      2          (deptno, dname, loc)
      3 VALUES  (dept_deptno_seq.NEXTVAL, 'SUPPORT', 'NY');

1 ligne créée.
```

Explication : Cette requête insère un nouveau département dans la table *DEPT*. On utilise la séquence pour générer le numéro de département suivant.

Exemple :

```
SQL> INSERT INTO emp(empno,ename,hiredate,deptno)
      2 VALUES  (7777, 'GEROGES',SYSDATE,dept_deptno_seq.CURRVAL);

1 ligne créée.
```

Explication : Cette requête insère un nouvel employé dans la table *EMP*.

Noter que les séquences qui sont en cache permettent un accès rapide.

### 5.1.6. Modifier une séquence

Vous pouvez modifier une séquence en utilisant **ALTER SEQUENCE**

Vous pouvez modifier la valeur d'incrément, la valeur maximale, la valeur minimale, les options cache et cycle.

```
ALTER SEQUENCE      nomsequence  
    [INCREMENT BY n]  
    [MAXVALUE n | NOMAXVALUE]  
    [MINVALUE n | NOMINVALUE]  
    [CYCLE | NOCYCLE]  
    [CACHE n | NOCACHE]
```

L'option **START WITH** ne peut être changé, qu'en recréant la séquence.  
La nouvelle valeur **MAXVALUE** ne peut pas être inférieure à la valeur précédente

Exemple :

```
SQL> ALTER SEQUENCE      dept_deptno_seq  
      2 MAXVALUE          30;  
  
ALTER SEQUENCE dept_deptno_seq  
*  
ERREUR à la ligne 1 :  
ORA-04005: INCREMENT doit être inférieur e la différence entre  
MAXVALUE et MINVALUE
```

Explication : La valeur **MAXVALUE** ne peut pas être changée.

### 5.1.7. Supprimer une séquence

Pour supprimer une séquence, il faut utiliser **DROP SEQUENCE**

```
DROP SEQUENCE nomsequence;
```

Exemple:

```
SQL> DROP SEQUENCE dept_deptno_seq;  
  
Séquence supprimée.
```

## 5.2. Index

### 5.2.1. Qu'est ce qu'un index

Un index est un objet qui peut augmenter la vitesse de récupération des lignes en utilisant les pointeurs.

Les index peuvent être créés automatiquement par le serveur Oracle ou manuellement par l'utilisateur. Ils sont indépendants donc lorsque vous supprimez ou modifiez un index les tables ne sont pas affectées.

Mais lorsque vous supprimez une table indexée, les index sont automatiquement supprimés. Lors de la création des clés primaires ou des clés étrangères, les index sont créés automatiquement et ils possèdent les même noms que les contraintes.

### 5.2.2. Création d'un index

Pour créer un index, il faut utiliser la requête **CREATE INDEX**.

```
CREATE INDEX   nomindex
ON           table (column1, column2 ...);
```

#### Quand créer un index:

- La colonne contient une large plage de valeurs.
- La colonne contient plusieurs valeurs nulles.
- Une ou plusieurs colonnes sont fréquemment utilisées dans la clause **WHERE** ou pour les conditions de jointure.
- La table est grande et la plupart des requêtes recherchent moins de 2-4% des lignes.

#### Quand ne pas créer des index:

- La table est petite.
- Les colonnes ne sont pas souvent utilisées.
- Les requêtes recherchent plus de 2-4% des lignes.
- La table est mise à jour fréquemment.
- Les colonnes indexées sont référencées comme une partie de l'expression.

Exemple :

```
SQL> CREATE INDEX      emp_ename_idx
      2 ON              emp (ename) ;

Index créé.
```

### 5.2.3. Vérification des index

Pour vérifier l'existence d'un index, vous pouvez interroger la table **USER\_INDEXES**.

Exemple :

```
SQL> SELECT      cix.index_name, cix.column_name,
      2          cix.column_position, uix.uniqueness
      3 FROM      user_indexes uix, user_ind_columns cix
      4 WHERE     cix.index_name = uix.index_name
      5 AND      cix.table_name='EMP' ;
```

INDEX_NAME	COLUMN_NAME	COLUMN_POSITION	UNIQUENES
EMP_ENAME_IDX	ENAME	1	NONUNIQUE



Exemple :

```
SQL> CREATE SYNONYM      emplo
      2 FOR                emp;

Synonyme créée.
```

Vous pouvez utiliser *EMPLO* pour la requête suivante:

```
SQL> SELECT      ename
      2 FROM        emplo;

      ENAME
      -----
      SMITH
      ALLEN
      WARD
      JONES
      MARTIN
      BLAKE
      CLARK
      TURNER
      ADAMS
      ...
      15 lignes sélectionnées.
```

### 5.3.3. Supprimer un synonyme

Pour supprimer un synonyme utilisez la requête **DROP SYNONYM**

```
DROP SYNONYM  nomsynonyme;
```

Exemple :

```
SQL> DROP SYNONYM      emplo;

Synonyme supprimé.
```

## 6. CONTROLE D'ACCES DES UTILISATEURS

### 6.1. Les privilèges système

#### 6.1.1. Qu'est ce qu'un privilège

Les privilèges sont des droits pour exécuter des requêtes

Le plus haut niveau de privilèges sont des privilèges DBA, il a la possibilité de donner aux utilisateurs l'accès à la base de données.

Les utilisateurs doivent posséder des privilèges système pour se connecter à la base de données, et les privilèges objets pour manipuler des données.

#### 6.1.2. Les privilèges DBA

Il existe plus de mille privilèges pour les utilisateurs et les rôles

Privilège système	Opération autorisée
<b>CREATE USER</b>	Autorise de créer des utilisateurs
<b>DROP USER</b>	Autorise de supprimer des utilisateurs
<b>DROP ANY TABLE</b>	Autorise de supprimer toutes les tables dans tous les schémas
<b>BACKUP ANY TABLE</b>	Autorise de sauvegarder toutes les tables dans tous les schémas.
<b>SELECT ANY TABLE</b>	Autorise d'effectuer les requêtes <b>SELECT</b> dans tous les schémas.
<b>CREATE ANY TABLE</b>	Autorise de créer des tables dans tous les schémas.

#### 6.1.3. Créer un utilisateur

Un DBA peut créer des utilisateurs en utilisant la requête **CREATE USER**.

Lorsqu'un utilisateur est créé, il ne possède aucun privilège. Le DBA doit lui donner des privilèges souhaités.

```
CREATE USER      utilisateur  
IDENTIFIED BY  motdepasse;
```

#### 6.1.4. Les privilèges système accordés à un utilisateur

Lorsque le DBA a créé un utilisateur, il lui donne des privilèges.

Exemple : Le DBA donne à l'utilisateur la possibilité de se connecter à la base de données. Ce privilège est donné grâce à **CREATE SESSION**.

```
GRANT          privilege [,privilege, ... ]  
TO            user [, user, role, PUBLIC ...];
```

Privilège	Opération autorisée
<b>CREATE SESSION</b>	Autorise à se connecter sur la base de données
<b>CREATE TABLE</b>	Autorise à créer des tables
<b>CREATE SEQUENCE</b>	Autorise à créer des séquences
<b>CREATE VIEW</b>	Autorise à créer des vues
<b>CREATE PROCEDURE</b>	Autorise à créer des procédures, des fonctions ou des packages

### 6.1.5. Accorder un privilège

Pour accorder un privilège il faut suivre les étapes suivantes :

- Créer un nouvel utilisateur.
- Donner le privilège **CREATE SESSION** à l'utilisateur
- Donner au nouvel utilisateur le privilège **CREATE TABLE**.

Exemple :

```
SQL> CREATE USER          scott2
  2 IDENTIFIED BY        tiger2;

Utilisateur crée.

SQL> GRANT CREATE SESSION
  2 TO                   scott2;

Autorisation de privilèges (GRANT) acceptée.

SQL> GRANT CREATE TABLE
  2 TO SCOTT2;

Autorisation de privilèges (GRANT) acceptée.
```

### 6.1.6. Créer et accorder un privilège à un rôle

Un rôle est un ensemble de privilèges

**CREATE ROLE** *nomrole*;

Lorsque le rôle est créé le DBA vous devez utiliser la requête **GRANT** pour assigner ce rôle aux utilisateurs.

Exemple:

```
SQL> CREATE ROLE          manager;

Rôle crée.

SQL> GRANT CREATE TABLE, SELECT ANY TABLE
  2 TO                   manager;

Autorisation de privilèges (GRANT) acceptée.
```



Explication : Cette requête crée le rôle **MANAGER** et donne à tous les utilisateurs la possibilité de créer les tables et sélectionner des données depuis toutes les tables de tous les schémas.

### 6.1.7. Modification de mot de passe

Pour changer le mot de passe, vous devez utiliser la requête **ALTER USER**

```
ALTER USER      utilisateur
IDENTIFIED BY nouveaupassword;
```

Exemple :

```
SQL> ALTER USER      scott2
      2 IDENTIFIED BY  oracle;

Utilisateur modifié.
```

Explication : On attribue le nouveau mot de passe **ORACLE** à **SCOTT2**

## 6.2. Les privilèges objet

### 6.2.1. Les privilèges objet

Un privilège objet donne le droit d'effectuer des opérations sur des tables, vues, séquences ou procédures spécifiques.

Voici le tableau des privilèges pour différents objets.

Privilège objet	Table	Vue	Séquence	Procédure
<b>ALTER</b>	X		X	
<b>DELETE</b>	X	X		
<b>EXECUTE</b>				X
<b>INDEX</b>	X			
<b>INSERT</b>	X	X		
<b>REFERENCES</b>	X	X		
<b>SELECT</b>	X	X	X	
<b>UPDATE</b>	X	X		

Les privilèges objet varient d'objet à objet.

Notez que le propriétaire d'un objet possède tous les privilèges sur cet objet

### 6.2.2. Accorder les privilèges

Le propriétaire d'un objet peut donner n'importe quel privilège objet à un autre utilisateur grâce à la requête **GRANT**

Si cette requête possède l'option **WITH GRANT OPTION**, le nouvel utilisateur pourra donner le privilège objet à un autre utilisateur

```

GRANT           privilege [(column)]
ON             objectname
TO            username | role | PUBLIC
[WITH GRANT OPTION];

```

Privilege:	Nom de privilège
Column:	Spécifie la colonne de la table ou la vue.
Objectname:	Le nom de l'objet sur lequel le privilège sera accordé.
Username:	Le nom du nouvel utilisateur.
Pubic:	Spécifie que le privilège est accordé a tous les utilisateurs.
WITH GRANT OPTION:	Donne au nouvel utilisateur la possibilité d'accorder les privilèges sur cet objet.

Exemple :

```

SQL> GRANT UPDATE
2 ON      emp
3 TO      scott;

Autorisation de privilèges (GRANT) acceptée.

```

Explication : Cette requête donne à utilisateur SCOOT la possibilité de mettre à jour la table *EMP*

### 6.2.3. Utilisation des mots clés WITH GRANT OPTION et PUBLIC

#### Mot clé WITH GRANT OPTION

Le privilège accordé avec la clause **WITH GRANT OPTION** donne la possibilité au nouvel utilisateur d'accorder les privilèges sur cet objet aux autres utilisateurs  
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.

Exemple :

```

SQL> GRANT          SELECT, UPDATE
2 ON      dept
3 TO      scott
4 WITH GRANT OPTION;

Autorisation de privilèges (GRANT) acceptée.

```

Explication : Cette requête donne à l'utilisateur SCOTT l'accès à la table *DEPT* avec des privilèges de sélectionner et de mettre à jour les données. SCOTT pourra accorder des privilèges aux autres utilisateurs.

#### Le mot clé PUBLIC:

Le possesseur de la table peut donner accès à tous les utilisateurs sur sa table grâce au mot clé **PUBLIC**.

Exemple :

```
SQL> GRANT          SELECT
      2 ON            emp
      3 TO PUBLIC;
```

Autorisation de privilèges (GRANT) acceptée.

### 6.2.4. Confirmation des privilèges accordés

Vous pouvez accéder au dictionnaire de données pour voir les privilèges que vous avez accordés

Dictionnaire de données	Description
<b>ROLE_SYS_PRIVS</b>	Les privilèges système donnés à un rôle.
<b>ROLE_TAB_PRIVS</b>	Les privilèges sur les tables donnés à un rôle.
<b>USER_ROLE_PRIVS</b>	Rôles accessibles par l'utilisateur.
<b>USER_TAB_PRIVS_MADE</b>	Les privilèges objet accordés à l'objet de l'utilisateur.
<b>USER_TAB_PRIVS_RECD</b>	Les privilèges objet accordés à l'utilisateur.
<b>USER_COL_PRIVS_MADE</b>	Les privilèges objet accordés sur des colonnes appartenant à l'utilisateur
<b>USER_COL_PRIVS_RECD</b>	Les privilèges objet que possède un utilisateur sur des colonnes
<b>USER_SYS_PRIVS</b>	List des privilèges système accordés à l'utilisateur.

### 6.2.5. Retirer les privilèges

Vous pouvez retirer des privilèges en utilisant la requête REVOKE

```
REVOKE                privilege[,..., ALL]
ON                   objectname
FROM                 user | role | PUBLIC;
[CASCADE CONSTRAINTS]
```

**CASCADE CONSTRAINTS:** Supprime toutes les contraintes d'intégrité référencée

Exemple :

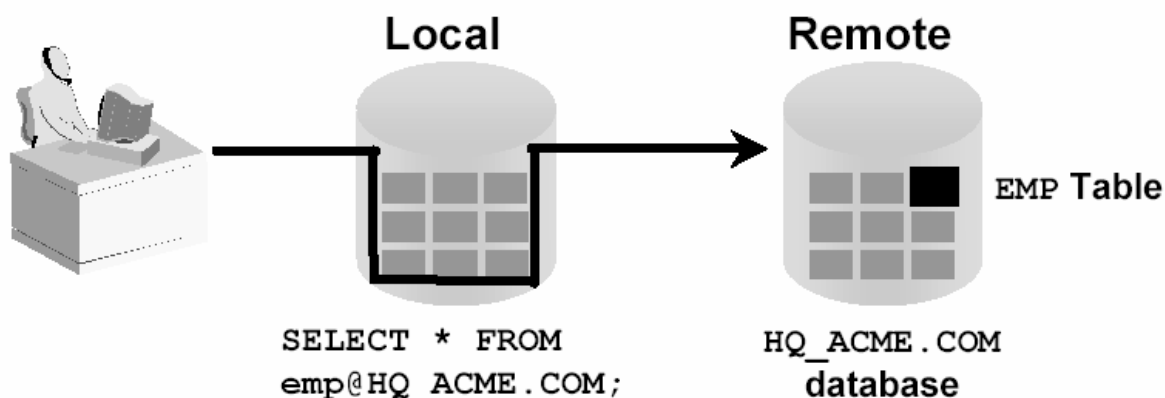
```
SQL> REVOKE          SELECT
      2 ON            emp
      3 FROM          scott;
```

Suppression de privilèges (REVOKE) acceptée.

Explication : Cette requête retire la possibilité de sélectionner des données de la table *EMP* pour l'utilisateur SCOTT.

## 6.3. Les liens symbolique de base de données

Les liens permettent aux utilisateurs d'accéder à la base de données distante.



Un lien est un pointeur qui définit le chemin de communication à sens unique depuis le serveur Oracle à un autre serveur

C'est à dire que le client, connecté à la base de données locale **A**, peut utiliser le lien stocké dans la base de données **A** pour accéder à la base de données distante **B**, mais les utilisateurs connectés à la base de données **B** ne peuvent pas utiliser ce lien pour accéder à la base de données **A**.

Si les utilisateurs de la base de données **B** veulent accéder aux informations de la base de données **A**, il faut qu'ils créent un lien.

L'avantage de ces liens est que les utilisateurs peuvent accéder à l'objet de l'utilisateur de la base de donnée distante tout en étant limités par des privilèges définis par le propriétaire de l'objet.

Noter que le dictionnaire de données USER\_DB\_LINKS contient des informations sur des liens accessibles par l'utilisateur.

Exemple :

```
SQL> DESCRIBE USER_DB_LINKS;
```

Nom	NULL ?	Type
DB_LINK	NOT NULL	VARCHAR2(128)
USERNAME		VARCHAR2(30)
PASSWORD		VARCHAR2(30)
HOST		VARCHAR2(2000)
CREATED	NOT NULL	DATE

Vous pouvez créer un lien en utilisant **CREATE DATABASE LINK**.

Exemple :

```
SQL> CREATE PUBLIC DATABASE LINK hq.acme.com
2 USING 'sales';
Lien de base de données créé.
```