

Le langage SQL

Pour Oracle

Auteur : Clotilde Garde

Société : Tellora

Version 2

Du 21 Novembre 2015

# Sommaire

<b>1</b>	<b>Les bases de données relationnelles (SGBDRO)</b>	<b>6</b>
<b>2</b>	<b>La base de donnée Oracle et ses outils</b>	<b>7</b>
2.1	L'outil SQL*Plus	7
2.1.1	<b>Environnement de travail</b>	<b>7</b>
2.1.2	<b>Lancement de SQL*Plus sous Dos</b>	<b>9</b>
2.2	L'outil iSQL*Plus	10
2.3	Outil Oracle Database Control ou Grid Control	10
<b>3</b>	<b>Les commandes de l'outil SQL*Plus</b>	<b>12</b>
3.1	Quelques commandes SQL*Plus	12
3.1.1	<b>Mise en forme à l'affichage</b>	<b>12</b>
3.1.2	<b>Ajouter des commentaires</b>	<b>13</b>
3.1.3	<b>Exécuter le contenu d'un script</b>	<b>13</b>
3.2	Utilisation de paramètres	14
3.2.1	<b>Déclarer un éditeur</b>	<b>14</b>
3.2.2	<b>Générer un fichier résultat</b>	<b>14</b>
3.2.3	<b>Modifier l'affichage par défaut</b>	<b>15</b>
3.2.4	<b>Mesurer les performances d'une requête</b>	Erreur ! Signet non défini.
<b>4</b>	<b>La base exemple</b>	<b>16</b>
4.1	Le MCD (Modèle Conceptuel de Données)	16
4.2	Règles de passage du MCD au MLD	17
1.1	Diagramme de classe	<b>Erreur ! Signet non défini.</b>
2.1	Du diagramme de classe au MLD	<b>Erreur ! Signet non défini.</b>
4.3	Le MLD (Modèle Logique de Données)	18
4.4	Les contraintes d'intégrité	18
4.5	Script de création des tables	19
4.6	Les types de données	22
4.7	Règles de nommage	23
<b>5</b>	<b>La commande SQL « CREATE TABLE »</b>	<b>24</b>
5.1	Modifier la structure d'une table	25
5.2	Contraintes d'intégrité activées ou désactivées	27
5.3	Contraintes immédiates ou différées	28
5.4	Manipulation des LOB	29
5.5	Manipulation des BFILEs	30
5.6	Créer une table à partir d'une table existante	31
<b>6</b>	<b>Notion de tablespace</b>	<b>33</b>
<b>7</b>	<b>Le dictionnaire de données</b>	<b>34</b>
<b>8</b>	<b>Le langage SQL</b>	<b>36</b>



8.1	Requêtes avec comparaisons .....	39
8.1.1	<b>La clause IN</b> .....	<b>39</b>
8.1.2	<b>La clause LIKE</b> .....	<b>40</b>
8.1.3	<b>La valeur NULL</b> .....	<b>40</b>
8.1.4	<b>La clause BETWEEN</b> .....	<b>41</b>
8.1.5	<b>Trier l’affichage d’une requête</b> .....	<b>41</b>
8.1.6	<b>Eliminer les doublons</b> .....	<b>42</b>
8.2	Requêtes avec jointures .....	43
8.2.1	<b>Equijointure</b> .....	<b>43</b>
8.2.2	<b>Inequijointure</b> .....	<b>44</b>
8.2.3	<b>Jointure multiple</b> .....	<b>44</b>
8.2.4	<b>Utiliser des ALIAS</b> .....	<b>46</b>
8.2.5	<b>Auto-jointure</b> .....	<b>46</b>
8.2.6	<b>Jointure externe</b> .....	<b>47</b>
8.3	Ecriture et mise en forme .....	49
8.3.1	<b>Afficher un titre de colonne</b> .....	<b>49</b>
8.3.2	<b>Les opérateurs</b> .....	<b>50</b>
8.3.3	<b>Afficher un libellé dans une requête</b> .....	<b>50</b>
8.4	Les fonctions .....	51
8.4.1	<b>Les fonctions d’agrégat</b> .....	<b>51</b>
8.4.2	<b>Les fonctions numériques</b> .....	<b>52</b>
8.4.3	<b>Les fonctions de chaînes de caractères</b> .....	<b>52</b>
8.4.4	<b>Les fonctions de gestion du temps</b> .....	<b>55</b>
8.4.5	<b>Autres fonctions</b> .....	<b>57</b>
8.5	Requêtes avec regroupement.....	58
8.6	Requêtes ensemblistes .....	61
8.6.1	<b>Minus</b> .....	<b>62</b>
8.6.2	<b>UNION</b> .....	<b>62</b>
8.6.3	<b>INTERSECT</b> .....	<b>63</b>
8.7	Sous requêtes dans la clause FROM .....	63
8.8	Requêtes imbriquées .....	64
8.8.1	<b>Opérateurs de comparaison</b> .....	<b>64</b>
8.8.2	<b>Opérateurs ensemblistes</b> .....	<b>66</b>
8.9	Balayer une arborescence .....	68
<b>9</b>	<b>Les jointures ANSI</b> .....	<b>70</b>
3.1	Jointures simples .....	70
4.1	Jointure avec conditions.....	71
9.1.1	<b>L’opérateur JOIN ON</b> .....	<b>71</b>
9.1.2	<b>L’opérateur JOIN USING</b> .....	<b>71</b>
9.1.3	<b>L’opérateur NATURAL JOIN</b> .....	<b>72</b>
9.1.4	<b>Produit cartésien</b> .....	<b>73</b>
9.1.5	<b>Jointure externe</b> .....	<b>73</b>
<b>10</b>	<b>Transactions et accès concurrents</b> .....	<b>77</b>



10.1	Découper une transaction.....	78
10.2	Gestion des accès concurrents .....	79
10.3	Les verrous .....	80
10.4	Accès concurrents en mise à jours .....	81
10.5	Les rollbacks segments ou segments d'annulation.....	82
<b>11</b>	<b>Modifier les lignes de tables.....</b>	<b>83</b>
11.1	Insérer des lignes dans une table .....	83
11.1.1	<b>La commande INSERT.....</b>	<b>83</b>
11.1.2	<b>Insertion à partir d'une table existante .....</b>	<b>84</b>
11.2	Modifier les lignes d'une table .....	85
11.2.1	<b>La commande UPDATE .....</b>	<b>85</b>
11.2.2	<b>Modifications de lignes à partir d'une table existante.....</b>	<b>86</b>
11.3	Spécifier la valeur par défaut d'une colonne .....	87
11.4	Supprimer les lignes d'une table.....	88
11.4.1	<b>La commande DELETE.....</b>	<b>88</b>
11.4.2	<b>Vider une table .....</b>	<b>90</b>
<b>12</b>	<b>gestion de la confidentialité.....</b>	<b>91</b>
12.1	Gestion de la confidentialité niveau objet.....	92
12.2	Gestion de la confidentialité niveau system .....	94
12.3	Les rôles.....	95
<b>13</b>	<b>Notion de schéma .....</b>	<b>97</b>
13.1	Création d'un schéma .....	97
13.2	Intérêt d'un schéma .....	98
13.3	Modifier un élément de schéma.....	98
<b>14</b>	<b>Les objets de schema.....</b>	<b>100</b>
14.1	Les vues .....	100
14.1.1	<b>Créer une vue .....</b>	<b>100</b>
14.1.2	<b>Supprimer une vue .....</b>	<b>104</b>
14.2	Les synonymes .....	104
14.3	Les séquences .....	105
14.3.1	<b>Créer une séquence .....</b>	<b>106</b>
14.3.2	<b>Utiliser une séquence .....</b>	<b>106</b>
14.3.3	<b>Modifier une séquence.....</b>	<b>107</b>
14.3.4	<b>Supprimer une séquence .....</b>	<b>107</b>
14.4	Procédures, Fonctions et Packages .....	108
14.5	Les Triggers .....	109
14.6	Les index .....	110
14.6.1	<b>Index et contraintes d'intégrité .....</b>	<b>114</b>
14.6.2	<b>La clause USING INDEX .....</b>	<b>115</b>
14.6.3	<b>Suppression d'un index.....</b>	<b>116</b>
<b>15</b>	<b>Complément sur les tables.....</b>	<b>117</b>
15.1	Le Flach Back .....	117



<b>15.1.1</b>	<b>Modifier une table par fusion : MERGE .....</b>	<b>119</b>
<b>15.1.2</b>	<b>Améliorations de la commande MERGE en version 10g.....</b>	<b>122</b>
15.2	Créer une table à partir d'une table existante.....	123
15.3	Renommer une table .....	124
15.4	Les tables temporaires.....	124
15.5	Les tables externes .....	124
<b>16</b>	<b>Les vues Matérialisées .....</b>	<b>131</b>



# 1 Les bases de données relationnelles (SGBDRO)

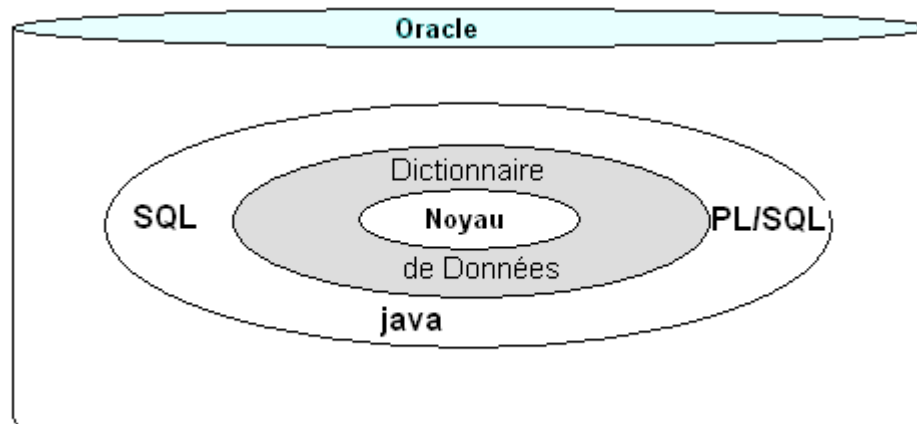
Les SBDRO (Système de Gestion de Bases de Données Relationnelles Objet) sont constituées de tables en relations les unes avec les autres.

Ces relations sont représentées et gérées par les contraintes d'intégrités au niveau du noyau de la base de données.

Ces contraintes d'intégrité sont vérifiées et exécutées tout au long de la vie de la base de données.

Elles garantissent :

- la lecture cohérente (même version des données au sein d'une même lecture)
- la cohérence des données (le respect de la conception de la base de données)



Le « O » de SGBDRO, représente l'intégration de l'objet à travers la gestion des images, de la vidéo, de la musique apparue avec l'arrivée d'internet.

Ces objets sont stockés dans les bases de données relationnelles dans des types BLOB, CLOB, Bitmap, ces types seront abordés dans les chapitres qui suivent.



## 2 La base de donnée Oracle et ses outils

Trois outils sont présents dans une base de données Oracle

SQL\*Plus (sqlplus), outil ligne de commande

iSQL\*Plus, outil graphique permettant de saisir des requêtes SQL.

Oracle Enterprise Manager (OEM), appelé encore Grid Control est un outil graphique d'administration.

### 2.1 L'outil SQL\*Plus

Outil ligne de commande nommé SQLPLUS.

Présent sur tous les serveurs de base de données Oracle.

```
SQLPLUS [ CONNEXION ] [ @FICHER_SCRIPT [ ARGUMENT [ ,... ] ] ]
```

Il permet de saisir et d'exécuter des ordres SQL ou du code PL/SQL et dispose en plus d'un certain nombre de commandes.

```
-- lancer SQLPlus sans connexion
C:\> SQLPLUS /NOLOG

-- lancer SQLPlus avec connexion
C:\> SQLPLUS system/tahiti@tahiti

-- visualiser l'utilisateur avec lequel on s'est connecté
SQL> show user
USER est "SYSTEM"

-- lancer SQLPlus avec connexion et lancement d'un script sur la ligne de commande
C:\> SQLPLUS system/tahiti@tahiti @info.sql
```

#### 2.1.1 Environnement de travail

SQL\*PLUS est avant tout un interpréteur de commandes SQL. Il est également fortement interfacé avec le système d'exploitation. Par exemple, sous UNIX, on pourra lancer des commandes UNIX sans quitter sa session SQL\*PLUS.

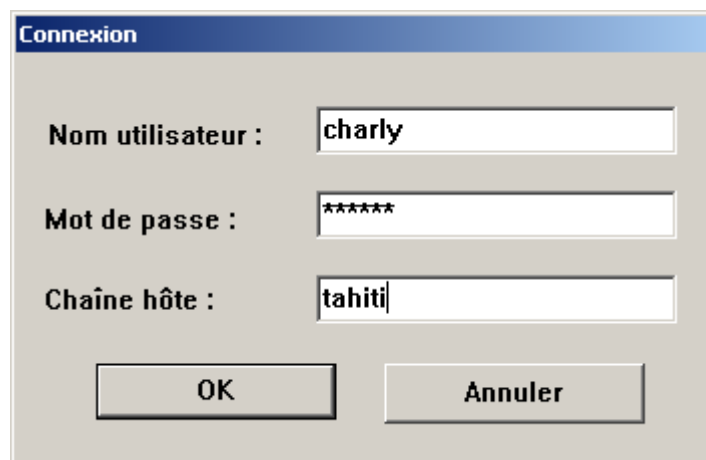
Un SGBDR est une application qui fonctionne sur un système d'exploitation donné. Par conséquent, il faut se connecter au système avant d'ouvrir une session ORACLE.



Connexion à une base de données Oracle en utilisant un client Oracle installé sur un poste distant.  
Depuis le groupe ORACLE, double sur l'icône SQL\*Plus ...



La boîte de dialogue suivante permet de saisir un compte et un mot de passe ORACLE ...



Connexion

Nom utilisateur :

Mot de passe :

Chaîne hôte :

Le nom de la « Chaîne hôte » correspond au nom du service Oracle Net de la base de données à laquelle l'utilisateur veut se connecter à distance.

La session SQL\*PLUS est ouverte ...

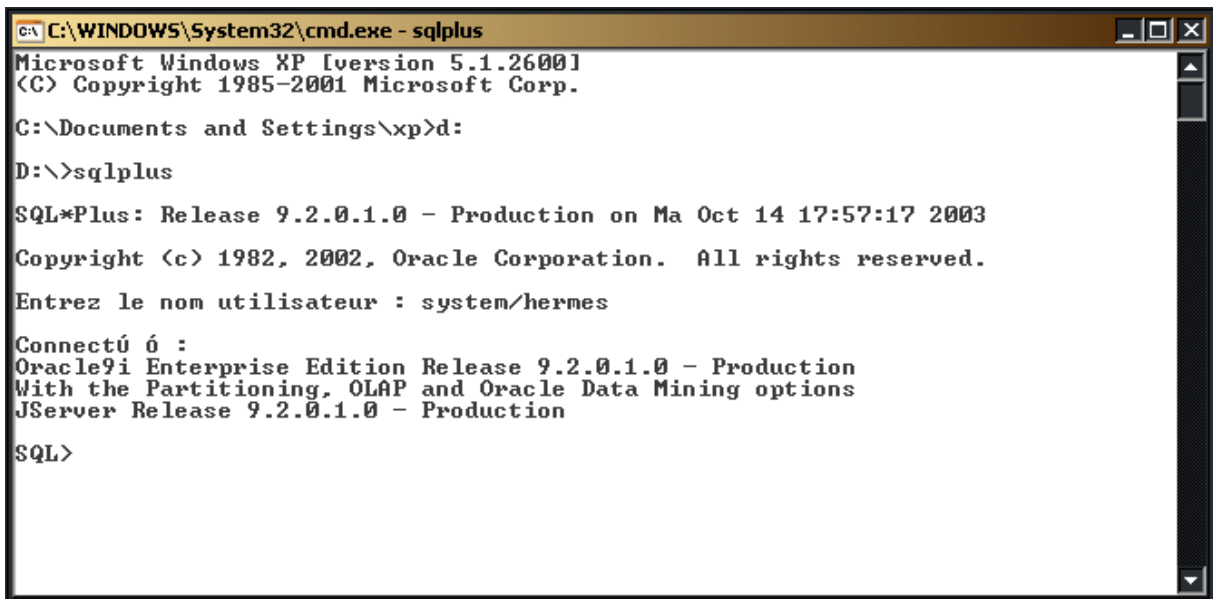




### 2.1.2 Lancement de SQL\*Plus sous Dos

Pour lancer SQL Plus sans se connecter à une base de données utilisez la commande :

Pour démarrer une session SQL Plus sous dos il suffit de se mettre en commande DOS puis d'exécuter la commande SQL PLUS .



```
C:\WINDOWS\System32\cmd.exe - sqlplus
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\xp>d:

D:\>sqlplus

SQL*Plus: Release 9.2.0.1.0 - Production on Ma Oct 14 17:57:17 2003
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Entrez le nom utilisateur : system/hermes

Connectú ó :
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL>
```

Vous pouvez également lancer SQL\*Plus sans vous connecter puis effectuer la connexion.

```
SQLPLUS /nolog
      Connect USER/MotdePasse@ServiceDistant
```

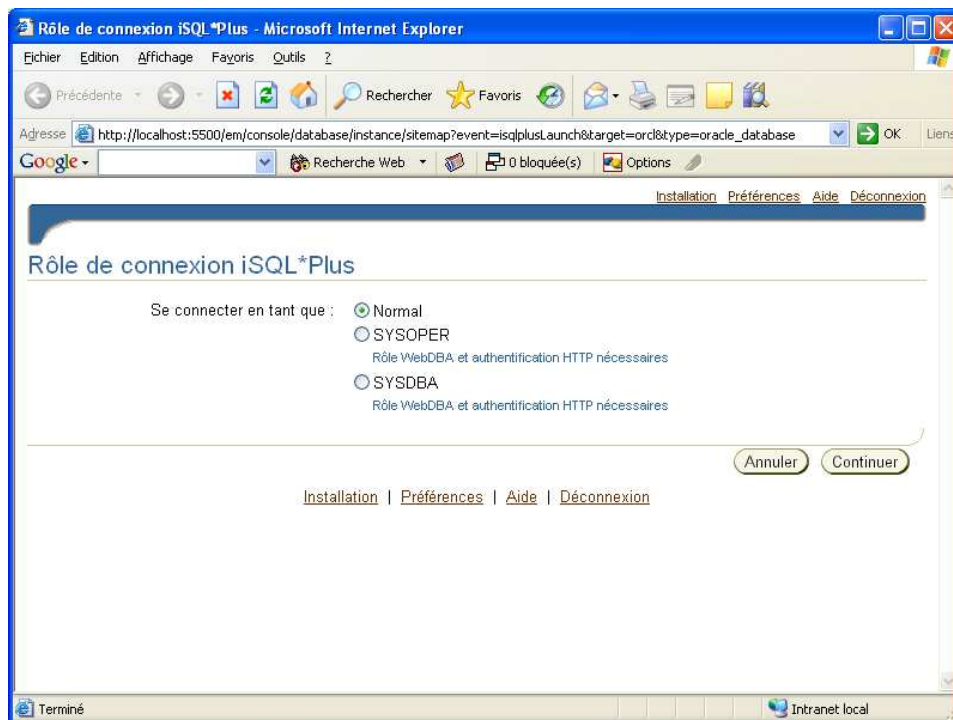
```
-- avec connexion
C:\> SQLPLUS charly/secret@tahiti

SQL> show user
USER est "charly"
SQL>
```



## 2.2 L'outil iSQL\*Plus

Outil Internet d'accès à une base de données Oracle.



## 2.3 Outil Oracle Database Control ou Grid Control

L'outil Oracle *Database Control* est une interface graphique permettant d'administrer une base de données unique.

Il contient un référentiel et est créé après la création de la base de données.



## Le langage SQL

Cette console permet d'administrer graphiquement la base de données :

- ◆ Arrêt/Démarrage, gestion du stockage, gestion des utilisateurs, gestion des schémas, ...
- ◆ Remontée d'alerte, de planification de tâche, de sauvegarde/restauration, d'export/import, ...

Le *Database Contrôle* est inclus dans l'installation standard.

Le *Grid Contrôle* est la console graphique vendue par Oracle, qui permet d'administrer un ensemble de bases de données sur des serveurs distants.

The screenshot shows the Oracle Enterprise Manager 10g Database Control interface for the BORA database instance. The browser window title is "Oracle Enterprise Manager (SYS) - Base de données : BORA - Microsoft Internet Explorer". The address bar shows the URL: http://localhost:5500/em/console/database/instance/sitemap?event=doLoad&target=BORA&type=oracle\_database.

The interface displays the following information:

- Statut:** Démarré (with an "Arrêter" button)
- Démarré depuis:** 20 avr. 2005 11:07:06
- Fuseau horaire:** CEST
- Disponibilité (%):** 100 (Dernières 24 heures)
- Nom de l'instance:** bora
- Version:** 10.1.0.2.0
- Read Only:** Non
- Répertoire d'origine Oracle Home:** D:\oracle\product\10.1.0
- Processus d'écoute:** LISTENER\_localhost
- Hôte:** localhost

**UC de l'hôte:** A bar chart showing CPU usage for "Autres" (blue) and "bora" (green). The "bora" instance is using approximately 15% of the CPU.

**Sessions actives:** A pie chart showing the distribution of sessions. The "E/S" (purple) category is the largest, followed by "UC" (green). The total number of active sessions is 0.16.

**Temps de réponse SQL (%):** Non disponible (comparé à la ligne de base)

**Haute disponibilité:** Temps de récupération de l'instance (secondes): 25; Dernière sauvegarde: n/a; Archivage: Non disponible; Nonne d'archivage utilisée (%): Non

**Utilisation de l'espace:** Taille de la base de données (Go): 0; Espaces disque logiques problématiques: 0 (indicated by a green checkmark); Résultats de la recherche des: Non configuré

**Récapitulatif des diagnostics:** Résultats de la recherche des performances: Aucune exécution ADDM n'est disponible; Toutes les violations: (indicated by a red warning icon)

The bottom of the screenshot shows the Windows taskbar with the system tray displaying "Intranet local" and the time "11:29".



### 3 Les commandes de l'outil *SQL\*Plus*

Les commandes SQL\*Plus sont des commandes de mise en forme pour la plupart liées à l'outil SQL\*Plus.

- A ne pas confondre avec des commandes SQL.

Une fois connecté l'utilisateur peut travailler sur la base de données.

Sous SQL\*Plus une requête SQL peut s'écrire sur plusieurs lignes. A chaque retour chariot l'interpréteur incrémente le numéro de ligne jusqu'au « ; » final qui marque la fin de la commande SQL.

```
SQL> select *
      2  from
      3  avion
      ;

      ID_AVION NOM_AVION
-----
          1 Caravelle
          2 Boeing
          3 Planeur
          4 A_Caravelle_2
```

#### 3.1 Quelques commandes SQL\*Plus

##### 3.1.1 *Mise en forme à l'affichage*

Les principales commandes de mise en forme sont présentées ci-dessous :

- ♦ SET LINESIZE 100, reformater la taille de la ligne à 100 caractères
- ♦ SET PAUSE ON, afficher un résultat page par page
- ♦ SET PAGESIZE 20 : affiche 20 lignes par page entre 2 entêtes de colonnes
- ♦ COL Nomcol FORMAT A20, formater l'affichage d'une colonne Nomcol sur 20 caractères
- ♦ COL Nomcol FORMAT 99.99, formater l'affichage d'une colonne Nomcol
- ♦ COL Nomcol FORMAT 0999999999, affiche le premier zéro
- ♦ CLEAR COL, ré-initialiser la taille des colonnes par défaut
- ♦ SHOW USER, visualiser le user sous lequel on est connecté
- ♦ CONNECT [User/MotPass@adresseServeur](#), changer de session utilisateur



- ◆ `CLEAR SCREEN`, ré-initialiser l'écran
- ◆ `SET SQLPROMPT TEST>` , affiche le prompt SQL en : `TEST>`
- ◆ `DESC NomTable`, afficher la structure d'une table ou d'une vue
- ◆ `/` , ré-active la dernière commande
- ◆ `SAVE NomFichier.txt [append|create|replace]` , permet de sauvegarder le contenu du buffer courant dans un fichier « `.sql` ».
- ◆ `TI ON|OFF`, provoque l'affichage de l'heure avec l'invite
- ◆ `SQL }` , spécifie le caractère « `}` » comme étant le caractère de continuation d'une commande SQL\*Plus.
- ◆ `SUFFIX txt`, spécifie l'extension par défaut des fichiers de commande SQL\*Plus

L'option `ON` permet d'activer la production de la ligne d'informations, `OFF` permet de la désactiver. La ligne d'information est système-dépendant.

### 3.1.2 Ajouter des commentaires

Le double tiret « `--` » ou la commande `REM` permettent d'ajouter des commentaires à une commande sur une seule ligne.

On peut saisir un commentaire multi-ligne en utilisant « `/* ... */` ».

### 3.1.3 Exécuter le contenu d'un script

Pour exécuter un ensemble de commandes dans un script SQL il suffit d'utiliser la commande `start nom_script` ou `@ nom_script`.

- ◆ `@ NomFichier.txt`, permet d'exécuter le contenu d'un fichier sql

```
SQL> start all_avion;

  ID_AVION NOM_AVION
-----
         1 Caravelle
         2 Boeing
         3 Planeur
         4 A_Caravelle_2
```

L'éditeur par défaut avec lequel s'interface SQL\*PLUS est le « Bloc-Notes » (`c:\Windows\notepad.exe`). Les principes précédents restent les mêmes.



## 3.2 Utilisation de paramètres

L'instruction `ACCEPT` permet de saisir des valeurs de paramètres (ce ne sont pas des variables et à ce titre ne nécessitent aucune déclaration).

```
ACCEPT reference NUMBER PROMPT 'Entrez la référence d'un avion: '
select * from avion where Id_avion=&reference;

SQL> @essai
Entrez la référence d'un avion: 1

  ID_AVION NOM_AVION
-----
         1 Caravelle
```

### 3.2.1 Déclarer un éditeur

Pour déclarer Notepad comme éditeur `SQL*PLUS`, et l'extension « .txt » pour exécuter un script il suffit de saisir ces deux lignes de commandes :

```
SET SUFFIX TXT
DEFINE _EDITOR = NOTPAD
```

Après avoir tapé ces 2 lignes de commandes taper :

- ◆ `ED` Pour afficher l'éditeur NotPad.

### 3.2.2 Générer un fichier résultat

La commande `SPOOL` permet de générer un fichier résultat contenant toutes les commandes passées à l'écran

- ◆ `SPOOL NomFichier.txt`, permet d'activer un fichier de format texte dans lequel on retrouvera les commandes et résultats affichés dans SQL Plus.
- ◆ `SPOOL OFF`, permet de désactiver le pool ouvert précédemment.

```
SPOOL MonFichier.txt
-- commandes SQL affichées
-- commandes SQL affichées
-- commandes SQL affichées

Spool OFF
```



### 3.2.3 *Modifier l'affichage par défaut*

L'affichage des commandes SQL lors de l'exécution d'un script peut être modifié par un ensemble de commandes :

- ◆ `SET PAUSE ON`, afficher un résultat page par page
- ◆ `SET LINESIZE 100`, reformater la taille de la ligne à 100 caractères
- ◆ `SET ECHO ON/OFF`, affiche ou pas le texte de la requête ou de la commande à exécuter
- ◆ `SET TIMING ON|OFF`, provoque l'affichage d'informations sur le temps écoulé, le nombre d'E/S après chaque requête
- ◆ `TERM [ON|OFF]`, supprime tout l'affichage sur le terminal lors de l'exécution d'un fichier
- ◆ `VER [ON|OFF]`, provoque l'affichage des lignes de commandes avant et après chaque substitution de paramètre.

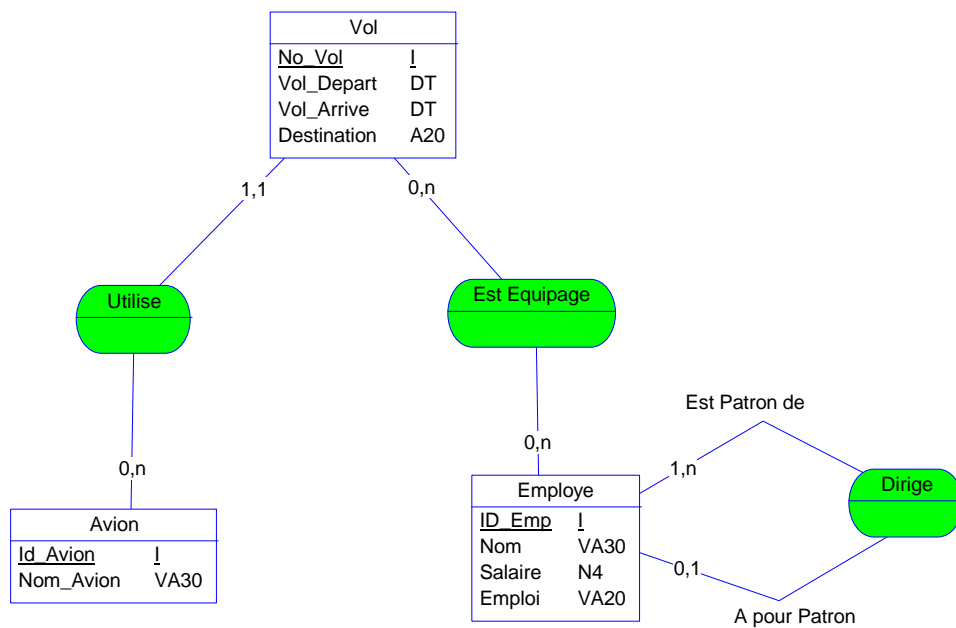


## 4 La base exemple

Nous vous présentons la base de données TAHITI qui servira de support aux exemples présentés dans le cours

### 4.1 Le MCD (Modèle Conceptuel de Données)

Modèle Conceptuel de Données		
Projet : Tahiti		
Modèle : Tahiti		
Auteur : Clotilde Attouche	Version	22/08/2004





## 4.2 Règles de passage du MCD au MLD

Le passage du Modèle Conceptuel de Données en Modèle Logique de données se fait en appliquant les règles citées ci-dessous :

Les entités deviennent des tables

- Les identifiants des entités deviennent les clés primaires de ces tables

Les relations ternaires (toutes les cardinalités sont 0,N ou 1,N de chaque côté de la relation) deviennent des tables

- La concaténation des identifiants des entités qui concourent à la relation devient la clé primaire de la table issue de la relation ; chacun, pris séparément, devient clé étrangère.

Les relations possédant des cardinalités 0,1 ou 1,1 d'un coté et 0,N de l'autre coté, on ajoute la colonne de l'identifiant dans la table coté 0,N de la relation.

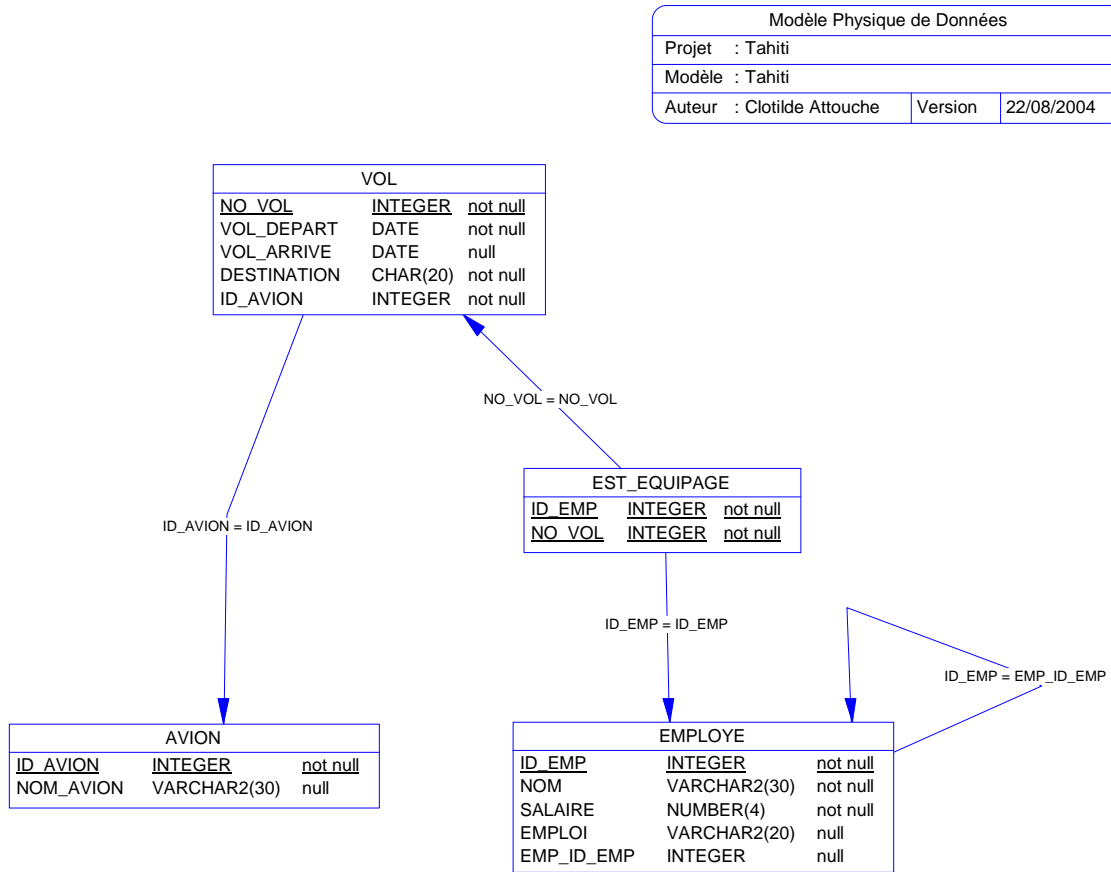
- Cette colonne devient clé étrangère et référence la clé primaire.

Pour les relations possédant des cardinalités 0,1 et 1,1 de chaque côté de la relation, il est préférable de créer une table, mais l'on peut également faire migrer l'identifiant dans l'une des deux entités ; celui ci devient alors clé étrangère (c'est ce que font des outils comme Power AMC)

- Power AMC est un outil permettant de concevoir des MCD, appelé AGL (Atelier de Génie Logiciel). Ces outils permettent de générer le script des tables automatiquement après la conception des MCD. Ces outils sont utilisés dans les projets de conception de nouveaux logiciels.



### 4.3 Le MLD (Modèle Logique de Données)



### 4.4 Les contraintes d'intégrité

Les contraintes d'intégrité garantissent l'intégrité des données. Elles se déclenchent automatiquement et sont gérées par Oracle au niveau du noyau de la base de données. Elles sont très rapides en exécution, beaucoup plus rapides qu'un programme externe au noyau de la base de données.

Les contraintes d'intégrité Oracle sont présentées ci-dessous :

- ◆ UNIQUE pour interdire les doublons sur le champ concerné,
- ◆ NOT NULL pour une valeur obligatoire du champ
- ◆ Clé primaire (PRIMARY KEY) pour l'identification des lignes (une valeur de clé primaire = une et une seule ligne),
- ◆ Clé étrangère (FOREIGN KEY) précisant que la valeur d'une colonne référence la valeur d'une colonne d'une autre table. Cette valeur doit exister auparavant dans cette autre table.
- ◆ CHECK pour préciser des domaines de valeurs.





Sous ORACLE, une clé primaire induit la création de deux contraintes (NOT NULL et UNIQUE), ainsi que la création d'un index !

## 4.5 Script de création des tables

Nous présentons le script de création des tables de la base de données « Tahiti » généré avec Power AMC.

- Ce script doit suivre exactement le modèle physique de données.

Le modèle physique de données est une version dégradée du modèle logique des données. Par exemple, redondance d'informations en vue d'optimisation applicative.

```
-- =====
-- Nom de la base      : TAHITI
-- Nom de SGBD        : ORACLE Version 8
-- Date de cr,ation   : 22/08/2004 17:09
-- =====

drop index EST_AFFECTE_PK
/

drop index EST_EQUIPAGE_FK
/

drop index EQUIPAGE_FK
/

drop table EST_EQUIPAGE cascade constraints
/

drop index VOL_PK
/

drop index UTILISE_FK
/

drop table VOL cascade constraints
/

drop index AVION_PK
/

drop table AVION cascade constraints
/
```



```

drop index EMPLOYE_PK
/

drop index A_POUR_PATRON_FK
/

drop table EMPLOYE cascade constraints
/

-- =====
-- Table : EMPLOYE
-- =====
create table EMPLOYE
(
  ID_EMP      INTEGER          not null,
  NOM         VARCHAR2(30)     not null,
  SALAIRE     NUMBER(4)        not null,
  EMPLOI      VARCHAR2(20)     null   ,
  EMP_ID_EMP  INTEGER          null   ,
  constraint PK_EMPLOYE primary key (ID_EMP)
    using index
    tablespace INDX
)
tablespace DATA
/

-- =====
-- Index : A_POUR_PATRON_FK
-- =====
create index A_POUR_PATRON_FK on EMPLOYE (EMP_ID_EMP asc)
tablespace INDX
/

-- =====
-- Table : AVION
-- =====
create table AVION
(
  ID_AVION    INTEGER          not null,
  NOM_AVION   VARCHAR2(30)     null   ,
  constraint PK_AVION primary key (ID_AVION)
    using index
    tablespace INDX
)
tablespace DATA
/

-- =====
-- Table : VOL
-- =====
create table VOL
(
  NO_VOL      INTEGER          not null,
  VOL_DEPART  DATE             not null,
  VOL_ARRIVE  DATE             null   ,
  DESTINATION CHAR(20)         not null,
  ID_AVION    INTEGER          not null,
  constraint PK_VOL primary key (NO_VOL)
    using index

```



```

        tablespace INDX
    )
tablespace DATA
/

-- =====
--   Index : UTILISE_FK
-- =====
create index UTILISE_FK on VOL (ID_AVION asc)
tablespace INDX
/

-- =====
--   Table : EST_EQUIPAGE
-- =====
create table EST_EQUIPAGE
(
    ID_EMP          INTEGER          not null,
    NO_VOL          INTEGER          not null,
    constraint PK_EST_EQUIPAGE primary key (ID_EMP, NO_VOL)
        using index
        tablespace INDX
)
tablespace DATA
/

-- =====
--   Index : EST_EQUIPAGE_FK
-- =====
create index EST_EQUIPAGE_FK on EST_EQUIPAGE (ID_EMP asc)
tablespace INDX
/

-- =====
--   Index : EQUIPAGE_FK
-- =====
create index EQUIPAGE_FK on EST_EQUIPAGE (NO_VOL asc)
tablespace INDX
/

-- =====
--   Index : CLES ETRANGERES
-- =====
alter table EMPLOYE
    add constraint FK_EMPLOYE_A_POUR_PA_EMPLOYE foreign key (EMP_ID_EMP)
        references EMPLOYE (ID_EMP)
/

alter table VOL
    add constraint FK_VOL_UTILISE_AVION foreign key (ID_AVION)
        references AVION (ID_AVION)
/

alter table EST_EQUIPAGE
    add constraint FK_EST_EQUI_EST_EQUIP_EMPLOYE foreign key (ID_EMP)
        references EMPLOYE (ID_EMP)
/

alter table EST_EQUIPAGE
    add constraint FK_EST_EQUI_EQUIPAGE_VOL foreign key (NO_VOL)

```



```

references VOL (NO_VOL)
/
alter table EMPLOYE
add CONSTRAINT SALAIRE_CC
CHECK (salaire >500);
/

```

## 4.6 Les types de données

Les différents types utilisés pour les colonnes de tables sont :

TYPE	VALEURS
BINARY-INTEGER	entiers allant de $-2^{31}$ à $2^{31}$ )
POSITIVE / NATURAL	entiers positifs allant jusqu'à $2^{31} - 1$
NUMBER	Numérique (entre $-2^{418}$ à $2^{418}$ )
INTEGER	Entier stocké en binaire (entre $-2^{126}$ à $2^{126}$ )
CHAR (n)	Chaîne fixe de 1 à 32767 caractères (différent pour une colonne de table)
VARCHAR2 (n)	Chaîne variable (1 à 32767 caractères)
LONG	idem VARCHAR2 (maximum 2 gigaoctets)
DATE	Date (ex. 01/01/1996 ou 01-01-1996 ou 01-JAN-96 ...)
CLOB	Grand objet caractère. Objets de type long stockés en binaire (maximum 4 giga octets) Déclare une variable gérant un pointeur sur un grand bloc de caractères, mono-octet et de longueur fixe, stocké en base de données.
BLOB	Grand objet binaire. Objets de type long (maximum 4 giga octets) Déclare une variable gérant un pointeur sur un grand objet binaire stocké dans la base de données (Son ou image).
NCLOB	Support en langage nationale (NLS) des grands objets caractères. Déclare une variable gérant un pointeur sur un grand bloc de caractères utilisant un jeu de caractères mono-octets, multi-octets de longueur fixe ou encore multi-octets de longueur variable et stocké en base de données.



ROWID	Composé de 6 octets binaires permettre d'identifier une ligne par son adresse physique dans la base de données.
UROWID	Le U de UROWID signifie Universel, une variable de ce type peut contenir n'importe quel type de ROWID de n'importe quel type de table.

Oracle vérifie la longueur maximum spécifiée lors de la déclaration des colonnes.

## 4.7 Règles de nommage

Un nom de structure Oracle doit respecter les règles suivantes

- ◆ 30 caractères maximums
- ◆ Doit commencer par une lettre
- ◆ Peut contenir des lettres, des chiffres et certains caractères spéciaux (\_\$#)
- ◆ N'est pas sensible à la casse
- ◆ Ne doit pas être un mot réservé Oracle



## 5 La commande SQL « CREATE TABLE »

Lors de la création de la table une contrainte peut être créée en fin de déclaration de table comme ci-dessous :

```
CREATE TABLE NOM_TABLE
(
  SPECIFICATION DES COLONNES
  SPECIFICATION DES CONTRAINTES D'INTEGRITE )
  SPECIFICATION DU STOCKAGE
;
```

### ◆ Spécification des colonnes

```
[NOM_COLONNE] TYPE [ DEFAULT 'VALEUR' ]
                [ NULL | NON_NULL ]
                [ , ... ]
```

### ◆ Contraintes d'intégrité

```
CONSTRAINT NOM_CONTRAINTE ]
{ PRIMARY KEY ( NOM_COLONNE [ , NOM_COLONNE ] ... )
  [ USING INDEX CLAUSE_INDEX ]

| UNIQUE ( NOM_COLONNE [ , NOM_COLONNE ] ... )
  [ USING INDEX CLAUSE_INDEX ]

| FOREIGN KEY ( NOM_COLONNE [ , NOM_COLONNE ] ... )
  REFERENCES [ SCHEMA. ] TABLE [ ( NOM_COLONNE ) ]
  [ ON DELETE CASCADE ]

| CHECK ( REGLE_CONDITIONS ) }
```





```
[ NOT DEFERRABLE | DEFERRABLE
      ( INITIALLY { IMMEDIATE | DEFERRED } ) ]
[ DISABLE | ENABLE [ VALIDATE | NOVALIDATE ] ]
```

#### ◆ Stockage

```
[ TABLESPACE NOM_TABLESPACE ]
[ PARALLEL ]
[ PCTFREE VALEUR ]
[ PCTUSED VALEUR ]
```

#### Exemple

```
create table VOL
(
  NO_VOL      INTEGER          not null,
  VOL_DEPART  DATE             not null,
  VOL_ARRIVE  DATE             null,
  DESTINATION CHAR(20)         not null,
  ID_AVION    INTEGER          not null,
  constraint PK_VOL primary key (NO_VOL)
  using index
  tablespace INDX
);
```

## 5.1 Modifier la structure d'une table

La modification de la structure d'une table se fait en utilisant la commande `ALTER`.

Il est possible :

- D'ajouter une colonne
- De modifier la taille d'une colonne
- D'ajouter ou de supprimer une contrainte d'intégrité



## Ajouter une colonne

Mêmes types que la commande CREATE  
attribut NOT NULL interdit (la nouvelle colonne contient NULL !)

## Modifier la taille d'une colonne

compatibilité avec le contenu de la colonne : real/integer  
augmentation de la taille  
changement de type d'une colonne vide  
spécifier NULL  
diminuer la taille d'une colonne non vide (depuis la 9i, avant on ne pouvait diminuer qu'une colonne vide)

## Ajouter ou supprimer une contrainte d'intégrité.

```
ALTER TABLE employe
ADD (Libelle VARCHAR2(20))
MODIFY (emploi VARCHAR2(20) NOT NULL);
```

La modification de la colonne EMPLOI en NOT NULL est valide si elle ne contient aucune valeur NULL.  
Diminuer la largeur d'une colonne non vide est possible, dans la limite de la plus grande valeur stockée dans la colonne.

```
Considérons le cas suivant
SQL> desc employe
Nom                                NULL ?  Type
-----
ID_EMP                             NOT NULL NUMBER(38)
NOM                                 NOT NULL VARCHAR2(30)
SALAIRE                             NOT NULL NUMBER(4)
EMPLOI                              VARCHAR2(20)
EMP_ID_EMP                           NUMBER(38)

SQL> select max(length(emploi)) from employe;
MAX(LENGTH(EMPLOI))
-----
16

--Diminution de la largeur de la colonne avec Oracle9i
SQL> alter table employe modify(emploi varchar2(18));
Table modifiée.

SQL> desc employe
Nom                                NULL ?  Type
-----
ID_EMP                             NOT NULL NUMBER(38)
NOM                                 NOT NULL VARCHAR2(30)
SALAIRE                             NOT NULL NUMBER(4)
```



EMPLOI	VARCHAR2(18)
EMP_ID_EMP	NUMBER(38)

La vue USER\_TABLES permet de visualiser les tables que l'on a créé.

```
SQL> select TABLE_NAME, TABLESPACE_NAME, TEMPORARY
2 from user_tables
3 order by table_name ;
```

TABLE_NAME	TABLESPACE_NAME	T
AVION	DATA	N
AVION_2	TOOL	N
EMPLOYE	DATA	N
EST_EQUIPAGE	DATA	N
VOL	DATA	N

## 5.2 Contraintes d'intégrité activées ou désactivées

Une contrainte d'intégrité peut être dans l'un des états suivant :

DISABLE : désactivée,

NOVALIDATE ENABLED : non validée activée (contrainte forcée, données incohérentes)

VALIDATE ENABLE : validée activée

La table USER\_CONSTRAINTS permet de visualiser les contraintes que l'on a créées.

```
SQL> select table_name, constraint_name, constraint_type, status
2 from user_constraints
3 order by table_name ;
```

TABLE_NAME	CONSTRAINT_NAME	C	STATUS
AVION	SYS_C001527	C	ENABLED
AVION	PK_AVION	P	ENABLED
AVION_2	SYS_C001542	C	ENABLED
EMPLOYE	SYS_C001523	C	ENABLED
EMPLOYE	SYS_C001524	C	ENABLED
EMPLOYE	SYS_C001525	C	ENABLED
EMPLOYE	PK_EMPLOYE	P	ENABLED
EMPLOYE	FK_EMPLOYE_A_POUR_PA_EMPLOYE	R	ENABLED
EMPLOYE	SALAIRE_CC	C	ENABLED
EST_EQUIPAGE	SYS_C001534	C	ENABLED
EST_EQUIPAGE	SYS_C001535	C	ENABLED
EST_EQUIPAGE	PK_EST_EQUIPAGE	P	ENABLED
EST_EQUIPAGE	FK_EST_EQUI_EST_EQUIP_EMPLOYE	R	ENABLED
EST_EQUIPAGE	FK_EST_EQUI_EQUIPAGE_VOL	R	ENABLED
VOL	SYS_C001529	C	ENABLED
VOL	SYS_C001530	C	ENABLED



```
VOL          SYS_C001531          C ENABLED
VOL          SYS_C001532          C ENABLED
VOL          PK_VOL           P ENABLED
VOL          FK_VOL_UTILISE_AVION R ENABLED
```

```
20 ligne(s) sélectionné(s).
```

### 5.3 Contraintes immédiates ou différées

Les contraintes non différées ou `IMMEDIATE` sont appliquées à la fin de chaque ordre LMD.

Une violation de contrainte entraîne l'annulation de l'ordre .

Une contrainte définie comme `IMMEDIATE` ne peut pas être modifiée pour être appliquée à la fin de la transaction.

Les contraintes différées sont vérifiées seulement lors de la validation d'une transaction.



Si une violation de contrainte est détectée, la transaction est annulée .

Ces contraintes sont utiles pour garantir la cohérence des données.

Pour qu'une contrainte soit de type différée il faut la déclarer à sa création :

`INITIALLY IMMEDIATE` : elle doit fonctionner par défaut comme une contrainte `IMMEDIATE` sauf si elle est définie autrement de façon explicite

`INITIALLY DEFERRED` : elle est forcée par défaut à la fin de la transaction



Il est conseillé d'adopter une convention standard de nomination des contraintes

Il est possible de créer les tables sans les contraintes d'intégrité puis de rajouter celle-ci par une mise à jour de table ultérieure en utilisant la commande :

```
ALTER TABLE  NOM_TABLE  ADD  CONSTRAINT  NOM_CONTRAINTE
;

```



```

-- =====
--      CLES ETRANGERES et CHECK
-- =====
alter table EMPLOYE
  add constraint FK_EMPLOYE_A_POUR_PA_EMPLOYE foreign key  (EMP_ID_EMP)
    references EMPLOYE (ID_EMP)
/
alter table VOL
  add constraint FK_VOL_UTILISE_AVION foreign key  (ID_AVION)
    references AVION (ID_AVION)
/
alter table EMPLOYE
  add CONSTRAINT SALAIRE_CC
  CHECK (salaire >500);
/

```

Il est possible de désactiver les contraintes par la commande :

```

alter table [ schema. ] nom_table ]
  disable { constraint nom_contrainte | primary key | unique
(nom_colonne [ , nom_colonne ] ... ) }
  [ cascade ]
;

```

La commande doit être utilisée pour une contrainte d'intégrité, si une clé primaire ou une contrainte unique est désignée comme clé étrangère, utilisez le paramètre `CASCADE` pour désactiver la clé étrangère avant de désactiver la clé primaire ou la contrainte unique.

De la même façon il est possible d'activer des contraintes désactivées.

```

SQL> alter table employe disable constraint  fk_employe_a_pour_pa_employe;
Table modifiée.

SQL> alter table employe enable constraint  fk_employe_a_pour_pa_employe;
Table modifiée.

```

## 5.4 Manipulation des LOB

Le SQL permet certaines actions de gestion des LOB.

Le package `DBMS_LOB` permet d'interagir complètement avec les LOB.

Méthode d'utilisation :

- ◆ Création de la table contenant le type LOB, et insertion des lignes
- ◆ Déclaration et initialisation du *handle* dans le programme



- ◆ Exécution d'une requête `SELECT FOR UPDATE` sur la ligne contenant l'indicateur du LOB
- ◆ Manipulation du LOB avec le package `DBMS_LOB` en utilisant le *handle* comme une référence aux données.
- ◆ Commit

```
Create table EMP
(
  Empno number(4),
  Ename varchar2(30),
  Carriere CLOB,
  Photo_identite BLOB
)
LOB (photo_identite)
  Store as (tablespace tbs_image
           Initial 10 M
           Next 2M )
;
```

Lorsque plusieurs colonnes LOB sont créées, chaque colonne possède des caractéristiques de stockage particulières.

Pour de bonnes performances placez le contenu des colonnes LOB dans des tablespaces différents.

Dans l'exemple précédent plusieurs segments sont créés :

- ◆ Le segment de la table
- ◆ Le segment pour les données de la colonne `CARRIERE`
- ◆ Le segment pour les données de la colonne `PHOTO_IDENTITE`
- ◆ 2 segments d'index pour référencer les `CHUNKS` des colonnes LOB.

Un `CHUNK` est un nombre de blocks Oracle contigus permettant de stocker les octets d'informations.

Chaque instance de LOB est un ensemble de `CHUNKS` qui ont la même taille.



Il est recommandé de créer les types LOB dans des tablespaces dédiés..

## 5.5 Manipulation des BFILEs

Ce type permet d'utiliser des fichiers stockés à l'extérieur de la base de données en ne conservant dans celle-ci qu'un pointeur sur ce fichier.



Opérations SQL :

- ◆ Définition des objets de type `BFILE`
- ◆ Association des types `BFILE` avec les fichiers externes
- ◆ Gestion de la sécurité des `BFILES`

Les autres opérations sont possibles avec le package `DBMS_LOB` et les OCI.

Une donnée de type `BFILE` est en lecture seule. Le fichier doit exister et se trouver dans la directory spécifiée et le processeur Oracle doit posséder le droit de lire celui-ci.

Lorsque la donnée `BFILE` est supprimée, le fichier existe toujours.

Chaque attribut de type `BFILE` d'une ligne peut contenir la référence à un fichier différent.

La `DIRECTORY` permet de spécifier un alias référençant un chemin d'accès sur le serveur où sont stockées les fichiers référencés par les données de type `BFILE`.

Le privilège `READ` permet d'accéder aux fichiers qui sont dans la directory, sans ce privilège les fichiers externes référencés par le type `BFILE` qui sont dans la directory ne sont pas accessibles.

```
CREATE [OR REPLACE] DIRECTORY NOM_DERECTORY AS 'PATH_NAME
;

```

Le chemin spécifié peut ne pas exister à la création de la directory, mais il doit exister lors de son utilisation avec les fichiers `BFILE`.

```

Create or replace directory emp_dir
  As 'app/Oracle/LOB/emp' ;

Grant read on directory emp_dir to role_personnel ;

```

## 5.6 Créer une table à partir d'une table existante

La création d'une table peut se faire à partir d'une table existante en précisant la requête d'extraction des colonnes désirées.

```

SQL> create table AVION_BIS
2 (id_avion, nom_avion)
3 as
4 select id_avion, nom_avion
5 from avion;

Table cr  e.

```



```
SQL> select * from avion_bis;
```

```
  ID_AVION NOM_AVION
```

```
-----
```

```
  1 Caravelle
```

```
  2 Boeing
```

```
  3 Planeur
```

La suppression d'une table se fait en utilisant la commande DROP.

```
SQL> drop table avion_2 ;
```

```
Table supprimée.
```

```
SQL> drop table vol;
```

```
drop table vol
```

```
*
```

```
ERREUR Ó la ligne 1 :
```

```
ORA-02449: clés uniques/primaires de la table référencées par des clés  
étrangères
```

### Suppression forcée d'une table

```
SQL> drop table vol cascade constraints;
```



Le contenu des autres tables reste inchangé et est donc incohérent par rapport aux valeurs utilisées, notamment la colonne Id\_emp de la table EST\_EQUIPAGE dans notre exemple.



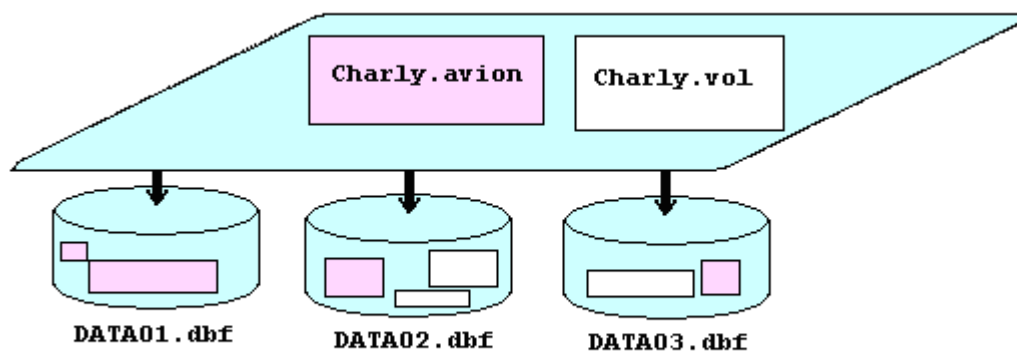


## 6 Notion de tablespace

Un tablespace est une unité logique de stockage composée d'un ou plusieurs fichiers physiques.

Les tablespaces sont des « conteneurs » dans lesquels sont stockés des tables ou des index qui utilisent de l'espace disque.

Ces tablespaces permettent à Oracle de faire le lien entre le logique et le physique et d'écrire sur le disque les données stockées dans les tables ou les index, c'est-à-dire les lignes des tables et les colonnes des index.



**Notion de Tablespace**

Ainsi les tablespaces sont représentés physiquement par des fichiers sur le disque.

Les tables ou les index sont appelés *segments* lorsqu'ils sont écrits dans les fichiers rattachés aux tablespaces.

```

create table VOL
(
  NO_VOL      INTEGER           not null,
  VOL_DEPART  DATE             not null,
  VOL_ARRIVE  DATE             null,
  DESTINATION CHAR(20)         not null,
  ID_AVION    INTEGER           not null,
  constraint PK_VOL primary key (NO_VOL)
  using index
  tablespace INDX
)
tablespace DATA
/

```



## 7 Le dictionnaire de données

C'est un ensemble de tables et de vues qui donnent des informations sur le contenu d'une base de données. Le dictionnaire de données est utilisé par Oracle pour traiter les requêtes.

Il contient :

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



Il appartient à l'utilisateur `SYS` et est stocké dans le tablespace `SYSTEM`.  
Sauf exception, toutes les informations sont stockées en `MAJUSCULE`.  
Il contient plus de 866 vues.

Il est créé lors de la création de la base de données, et mis à jour par Oracle lorsque des ordres `DDL` (*Data Définition Langage*) sont exécutés, par exemple `CREATE`, `ALTER`, `DROP` ...

Il est accessible en lecture par des ordres `SQL` (`SELECT`) et est composé de deux grands groupes de tables/vues :

Les tables et vues statiques

- ◆ Basées sur de vraies tables stockées dans le tablespace `SYSTEM`
- ◆ Accessible uniquement quand la base est ouverte « `OPEN` »
- ◆ Les tables et vues dynamiques de performance
- ◆ Ne sont en fait basées sur des informations en mémoire ou extraites du fichier de contrôle
- ◆ S'interrogent néanmoins comme de vraies tables/vues
- ◆ Donnent des informations sur le fonctionnement de la base, notamment sur les performances (d'où leur nom)
- ◆ Pour la plupart accessibles même lorsque la base n'est pas complètement ouverte (`MOUNT`)

Les vues statiques sont constituées de 3 catégories caractérisées par leur préfixe :

- ◆ `USER_*` : Informations sur les objets qui appartiennent à l'utilisateur
- ◆ `ALL_*` : Information sur les objets auxquels l'utilisateur a accès (les siens et ceux sur lesquels il a reçu des droits)
- ◆ `DBA_*` : Information sur tous les objets de la base



Derrière le préfixe, le reste du nom de la vue est représentatif de l'information accessible.

Les vues `DICTIONARY` et `DICT_COLUMNS` donnent la description de toutes les tables et vues du dictionnaire.

Oracle propose des synonymes sur certaines vues :

Synonyme	Vue correspondante
cols	User_tab_columns
dict	Dictionnary
ind	User_indexes
obj	User_objects
seq	User_sequences
syn	User_synonyms
tabs	User_tables

Les vues dynamiques de performance sont :

- ◆ Préfixées par « V\$ »
- ◆ Derrière le préfixe, le reste du nom de la vue est représentatif de l'information accessible
- ◆ Décrites dans les vues `DICTIONARY` et `DICT_COLUMNS`

Exemple de vues dynamiques

```
V$INSTANCE  
V$DATABASE  
V$SGA  
V$DATABASE  
V$PARAMETER
```



## 8 Le langage SQL

Le langage SQL (*Structured Query Language*) s'appuie sur les normes SQL ANSI en vigueur et est conforme à la norme SQL92 ou SQLV2 (ANSI X3.135-1889n, ISO Standard 9075, FIPS 127).

Il a été développé dans le milieu des années 1970 par IBM (*System R*). En 1979 Oracle Corporation est le premier à commercialiser un SGBD/R comprenant une incrémentation de SQL. Oracle comme acteur significatif intègre ses propres extensions aux ordres SQL.

Depuis l'arrivée d'internet et de l'objet Oracle fait évoluer la base de données et lui donne une orientation objet, on parle SGBDR/O : *System de Base de Données relationnel Objet*.

Les sous langages du SQL sont :

LID : Langage d'Interrogation des données, verbe SELECT

LMD : Langage de Manipulation des Données, utilisé pour la mise à jour des données, verbes INSERT, UPDATE, DELETE, COMMIT, ROLLBACK

LDD : Langage de définition des données, utilisé pour la définition et la manipulation d'objets tels que les tables, les vues, les index ..., verbe CREATE, ALTER, DROP, RENAME, TRUNCATE

LCD : Langage de Contrôle des Données, utilisé pour la gestion des autorisations et des privilèges, verbe GRANT, REVOKE

```
SELECT  LISTE DES COLONNES A AFFICHER (DANS L'ORDRE D'AFFICHAGE)
FROM    LISTE DES TABLES UTILISEES
WHERE   JOINTURE
        AND   CONDITION
;
```



```
SQL> connect charly/charly@tahiti
Connecté.
SQL> desc employe
Nom                                NULL ?  Type
-----
ID_EMP                             NOT NULL NUMBER(38)
NOM                                 NOT NULL VARCHAR2(30)
SALAIRE                             NOT NULL NUMBER(4)
EMPLOI                               VARCHAR2(18)
EMP_ID_EMP                           NUMBER(38)

SQL> select nom, salaire, emploi
2  from employe
3  where salaire >=2000
4  order by nom ;

NOM                                SALAIRE EMPLOI
-----
Marilyne                          2000 Hotesse de l'Air
Spirou                             2000 Pilote
```

- Affiche le nom, le salaire et l'emploi des employés dont le salaire est supérieur ou égal à 2000 Euros.

SELECT nom, salaire, emploi

WHERE salaire >= 2000

EMPLOYE				
ID_EMP	NOM	SALAIRE	EMPLOI	EMP_ID_EMP
1	Gaston	1700	Directeur	
2	Spirou	2000	Pilote	
3	Titeuf	1800	Stewart	2
4	Marilyne	2000	Hotesse de l'air	1

Comme on peut s'en rendre compte, une requête SELECT est très intuitive car elle se rapproche du langage quotidien.



C'est une des raisons du succès du SQL. Cependant, le SQL est avant tout un langage de définition et d'extraction de données. Ses possibilités algorithmiques, de saisie, ou d'affichage sont limitées. Ce n'est d'ailleurs pas sa finalité.

Lorsqu'il ne suffit plus (impossibilité syntaxique ou requête trop lourde), on utilise un autre langage qui offre une plus grande puissance algorithmique ou graphique.

Le SQL se contente alors d'extraire les données nécessaires pour le langage hôte (PL/SQL, Pro\*C, etc. ...). Beaucoup d'outils offrent en standard un interpréteur SQL pour consulter les données d'une base relationnelle (ORACLE ou autre).

Tous nos exemples vont s'appuyer sur la base exemple qui se présente dans l'état suivant :

```
SQL> select * from AVION;
```

ID_AVION	NOM_AVION
1	Caravelle
2	Boeing
3	Planeur
4	A_Caravelle_2

```
SQL> select * from VOL;
```

NO_VOL	VOL_DEPA	VOL_ARRI	DESTINATION	ID_AVION
1	04/09/04	05/09/04	Tahiti	1
2	09/09/04	10/09/04	Marquises	1
3	30/09/04		Tokyo	2

```
SQL> select * from EST_EQUIPAGE;
```

ID_EMP	NO_VOL
1	1
4	1
3	2
4	2

```
SQL> select * from EMPLOYE;
```

ID_EMP	NOM	SALAIRE	EMPLOI	EMP_ID_EMP
1	Gaston	1700	Directeur	
2	Spirou	2000	Pilote	1
3	Titeuf	1800	Stewart	2
4	Marilyne	2000	Hotesse de l'Air	1



Nous avons classé les différents types de requêtes par thème :

- Requêtes avec comparaisons
- Requêtes avec jointures
- Requêtes avec groupement
- Requêtes ensemblistes
- Sous requêtes
- Balayage d'une arborescence

## 8.1 Requêtes avec comparaisons

La clause `WHERE` peut utiliser les opérateurs :

`AND`, `OR`, `BETWEEN`, `NOT`, `IN`, `=`, `<>`, `!=`, `>`, `>=`, `<=`

Ces opérateurs s'appliquent aux valeurs numériques, aux chaînes de caractères, et aux dates. Les chaînes de caractères et les dates doivent être encadrées par `'...'` contrairement aux nombres.

```
SQL> select * from avion
  2  where id_avion between 1 and 2
  3    and nom_avion like 'C%' ;

ID_AVION NOM_AVION
-----
          1 Caravelle
```

### 8.1.1 La clause `IN`

La clause `IN` permet d'éviter l'emploi de `OR` et simplifie la syntaxe ...

```
SQL> select nom, salaire
  2  from employe
  3  where salaire in (1700,1800) ;

NOM                                SALAIRE
-----
Gaston                             1700
Titeuf                              1800
```

```
SQL> select nom, salaire
  2  from employe
  3  where salaire < 2000
  4  and emp_id_emp in (null, 2, 4)
  5  ;

NOM                                SALAIRE
-----
Gaston                             1700
Titeuf                              1800
```



### 8.1.2 La clause LIKE

La clause LIKE permet de rechercher des chaînes de caractères :

% Toute chaîne de 0 à n caractères  
 \_ 1 caractère

ESCAPE \ désigne \ pour inhiber les fonctions de « % » et « \_ »

```
SQL> select nom, salaire, emploi
  2   from employe
  3   where nom like '%t_n';
```

NOM	SALAIRE	EMPLOI
Gaston	1700	Directeur

```
SQL> select nom_avion
  2   from avion
  3   where nom_avion like '_\_%' escape '\';
```

```
NOM_AVION
-----
A_Caravelle_2
```

```
SQL> select nom_avion
  2   from avion
  3   where nom_avion like '*_%' escape '*';
```

```
NOM_AVION
-----
A_Caravelle_2
```

- Cette requête affiche tous les avions dont le nom commence par n'importe quel caractère suivi d'un \_
- Sans l'utilisation de % on se contenterait des noms sur 2 caractères qui respectent cette règle.

### 8.1.3 La valeur NULL

Pour manipuler une valeur non renseignée (en lecture ou mise à jour) on utilise le prédicat NULL



La valeur NULL pour un champ signifie non renseigné. Il ne faut pas confondre avec zéro ou blanc. Ce prédicat est utilisable dans toutes les commandes SQL (insert, select, ...).





```

SQL> select nom, emploi
      2  from employe
      3  where salaire is null ;

NOM                                EMPLOI
-----
Gaston                             Directeur

```

- Cette requête affiche le nom et le salaire des employés dont le salaire contient la valeur NULL

### 8.1.4 La clause BETWEEN

La clause Between permet de sélectionner des lignes à l'intérieure de bornes définies.

```

SQL> select * from avion
      2  where id_avion between 2 and 3 ;

ID_AVION NOM_AVION
-----
          2 Boeing
          3 Planeur

```

Cette requête affiche l'identifiant et le nom des avions dont l'identifiant est compris entre 2 et 3 bornes incluses.

### 8.1.5 Trier l'affichage d'une requête

La clause Order by permet de trier le résultat affiché.

```

SQL> select id_avion, nom_avion
      2  from avion
      3  order by nom_avion ;

ID_AVION NOM_AVION
-----
          4 A_Caravelle_2
          2 Bo'ng
          1 Caravelle
          3 Planeur

SQL> select id_avion, nom_avion
      2  from avion
      3  order by 2 ;

ID_AVION NOM_AVION
-----
          4 A_Caravelle_2
          2 Bo'ng
          1 Caravelle
          3 Planeur

```

- Cette requête affiche l'identifiant et le nom des avions ordonnés par nom d'avion, sur un ordre croissant.



Pour afficher un ordre décroissant il suffit de préciser Desc derrière la colonne citée dans le tri. Le nom de colonne peut être remplacé par la position de la colonne derrière la clause SELECT.

```
SQL> select id_avion, nom_avion
 2  from avion
 3  order by nom_avion desc ;

ID_AVION NOM_AVION
-----
      3 Planeur
      1 Caravelle
      2 Bo'ng
      4 A_Caravelle_2
```

### 8.1.6 *Eliminer les doublons*

Le mot clé DISTINCT permet d'éliminer les doublons lors de l'affichage. Il porte sur toutes les colonnes affichées sur une ligne.

```
SQL> select nom_avion, nom
 2  from employe, avion, est_equipage, vol
 3  where est_equipage.id_emp = employe.id_emp
 4  and est_equipage.no_vol = vol.no_vol
 5  and vol.id_avion = avion.id_avion
 6  and nom_avion = 'Caravelle'
 7  order by nom ;

NOM_AVION          NOM
-----
Caravelle          Gaston
Caravelle          Marilyne
Caravelle          Titeuf
Caravelle          Marilyne

SQL> select distinct nom_avion, nom
 2  from employe, avion, est_equipage, vol
 3  where est_equipage.id_emp = employe.id_emp
 4  and est_equipage.no_vol = vol.no_vol
 5  and vol.id_avion = avion.id_avion
 6  and nom_avion = 'Caravelle'
 7  order by nom ;

NOM_AVION          NOM
-----
Caravelle          Gaston
Caravelle          Marilyne
Caravelle          Titeuf
```

- Affichage des employés affectés à un équipage transportés par une caravelle.





DISTINCT provoque un tri,  
 a utiliser avec précautions.

## 8.2 Requêtes avec jointures

### Principe de base

Les requêtes concernent souvent des informations qui sont ventilées dans plusieurs tables. La recherche de ces informations s'appuie sur le principe de jointure. Il s'agit de rapprocher une ou plusieurs tables qui ont des colonnes en commun. Ces liens se traduisent la plupart du temps par des clés étrangères.

Une jointure est donc un sous ensemble du produit cartésien de deux tables. Seules les lignes respectant les conditions de jointures sont conservées. La différence réside dans la condition de jointure (`WHERE`) et dans les arguments du `SELECT`.

### 8.2.1 Equijointure

Nous souhaitons afficher le nom de tous les AVIONS qui sont utilisés pour un VOL.

Nous devons donc utiliser la table VOL pour lister tous les vols prévus, et la table AVION pour trouver le nom des avions. Mais il ne faut pas afficher le nom de tous les avions. Seuls ceux dont l'identifiant est mentionné dans la table VOL ont forcément été prévus pour voler.

Cette requête s'écrira :

```
SQL> select nom_avion Avion, destination
2   from vol, avion
3   where vol.id_avion = avion.id_avion
4   order by destination ;
```

La clause `FROM` doit préciser les tables concernées par la jointure.

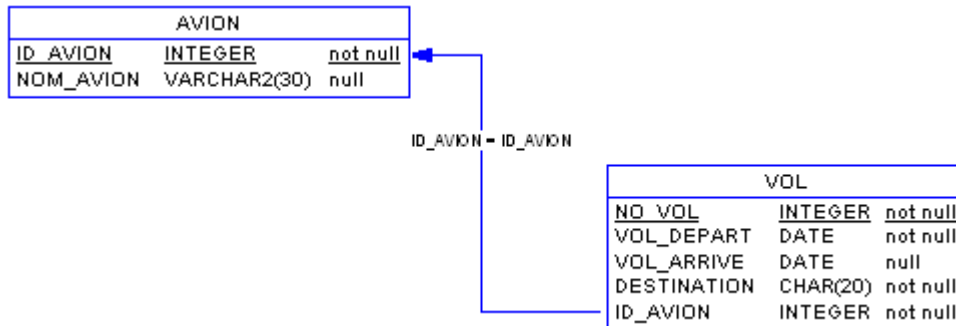
La clause `WHERE` doit préfixé la colonne `Id_avion` par le nom de la table concernée pour éviter les conflits.

En effet Oracle effectue d'abord le produit cartésien entre les tables « VOL » et « AVION » avant d'extraire les données à afficher .

La colonne `Id_avion` existe deux fois dans le produit cartésien et Oracle ne sait pas quelle colonne afficher.

Nous allons détailler cette requête afin de bien nous imprégner de l'algorithme de base mis en œuvre pour rechercher les données.





Tout se passe comme si l'interpréteur construisait une table temporaire résultant de toutes les associations possibles entre les lignes des deux tables.

Le système n'est pas capable de « deviner » les liens entre les deux tables. Il doit construire l'association des données des deux tables en s'appuyant sur les valeurs communes des champs *Id\_avion*.

Il suffit ensuite de ne garder que les lignes qui correspondent à la condition de jointure (ici égalité des champs *Id\_avion*) et d'afficher les informations demandées.

```
SQL> select nom_avion Avion, destination
2  from vol, avion
3  where vol.id_avion = avion.id_avion
4  order by destination ;
```

AVION	DESTINATION
Caravelle	Marquises
Caravelle	Tahiti
Boeing	Tokyo

Nous allons présenter maintenant d'autres types de jointures. Celui que nous venons de voir est une équi-jointure (la condition de jointure est une égalité sur deux colonnes de deux tables différentes).

### 8.2.2 Inequijointure

Une inéqui-jointure est une jointure sans condition d'égalité entre les deux colonnes.

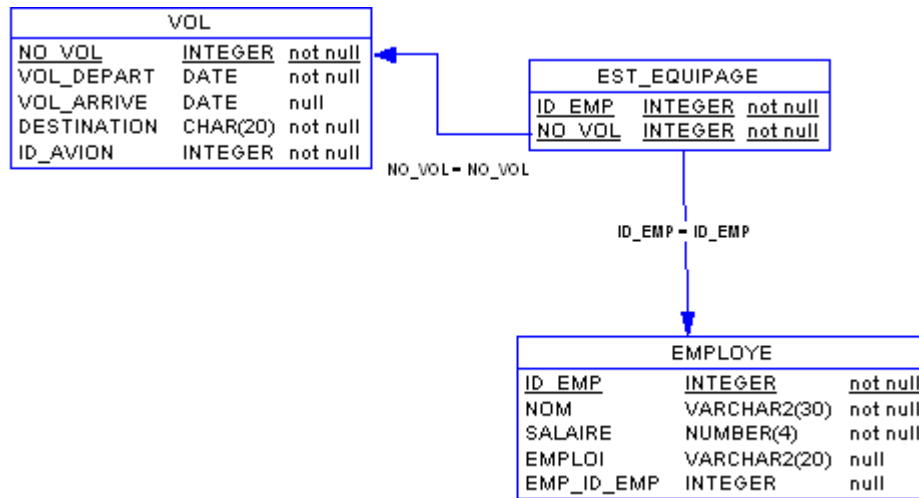
Elle utilise les opérateurs « <, >, <=, >=, <>, != » .

### 8.2.3 Jointure multiple

Une jointure multiple met en relation plusieurs colonnes de tables différentes, toujours en reliant ces tables par :

Clé étrangère vers clé primaire





Afficher les employés affectés à un vol pour Tahiti.

```

SQL> select nom, destination
2  from vol, employe, est_equipage
3  where est_equipage.id_emp = employe.id_emp
4        and est_equipage.no_vol = vol.no_vol
5        and destination = 'Tahiti' ;

NOM                                DESTINATION
-----
Gaston                             Tahiti
Marilyne                            Tahiti
    
```

Détail de la requête ...

Première jointure :

Sur la colonne *Id\_emp* entre les tables EMPLOYE et EST\_EQUIPAGE.

Seconde jointure :

Sur la colonne *No\_vol* entre les tables EST\_EQUIPAGE et VOL.

Condition :

La clause « and destination = 'Tahiti' » réduit la sélection concernant la destination.

Affichage :

La clause SELECT ne mentionnant que les colonnes « nom » et « destination » , Oracle n'affichera que ces deux colonnes.



### 8.2.4 Utiliser des ALIAS

Un alias permet de remplacer le nom d'une table dans un ordre select par une lettre. Le nom de la table n'est plus reconnu que par la lettre concernée dans la totalité de la requête.

Afficher l'équipage à destination de Tahiti.

```
SQL> select nom, destination
  2  from vol, employe, est_equipage
  3  where est_equipage.id_emp = employe.id_emp
  4         and est_equipage.no_vol = vol.no_vol
  5         and destination = 'Tahiti' ;
```

NOM	DESTINATION
Gaston	Tahiti
Marilyne	Tahiti

```
SQL> select e.id_emp, nom, destination
  2  from vol v, employe e, est_equipage eq
  3  where eq.id_emp = e.id_emp
  4         and eq.no_vol = v.no_vol
  5         and destination = 'Tahiti' ;
```

ID_EMP	NOM	DESTINATION
1	Gaston	Tahiti
4	Marilyne	Tahiti

### 8.2.5 Auto-jointure

Une auto-jointure est une jointure récursive sur une seule table. Si une table comporte n lignes, une auto-jointure sur cette table nécessitera au pire n x n comparaisons.

- ◆ Afficher les employés qui managent d'autres employés.

Dans le cas d'une auto-jointure, l'utilisation des alias est incontournable.

```
SQL> select distinct e1.nom
  2  from employe e1, employe e2
  3  where e1.id_emp = e2.emp_id_emp;
```

NOM
Gaston
Spirou



Vérification :

```
SQL> select nom, id_emp, emp_id_emp Manageur
2   from employe
3   order by nom;
```

NOM	ID_EMP	MANAGEUR
Gaston	1	
Marilyne	4	1
Spirou	2	1
Titeuf	3	2

L'auto-jointure est utilisée pour comparer la valeur d'une colonne dans une ligne de la table par rapport aux autres valeurs contenues dans les lignes de la même colonne et de la même table.

Pour réussir une auto-jointure il suffit d'imaginer que la base de donnée contient deux tables identiques portant des noms différents : « e1 » et « e2 » .

Comme on ne précise pas la jointure, Oracle effectue un produit cartésien entre la table et elle même.

### 8.2.6 Jointure externe

Nous allons présenter le principe de jointure externe par un exemple progressif.

Dans un premier temps, nous souhaitons afficher tous les employés (nom et N° du vol) qui sont affectés à un vol.

Cette requête peut s'exprimer :

```
SQL> select nom, no_vol
2   from employe e, est_equipage eq
3   where eq.id_emp = e.id_emp
4   order by nom ;
```

NOM	NO_VOL
Gaston	1
Marilyne	1
Marilyne	2
Titeuf	2

Les lignes provenant des tables `EMPLOYEE` et `EST_EQUIPAGE` partagent toutes la colonne `Id_emp` sur laquelle porte la jointure.

Seules les lignes qui vérifient la condition de jointure seront donc affichées.

Si l'on désire afficher tous les employés qui sont affectés ou non à un vol et les informations sur ce vol on utilisera une jointure externe.

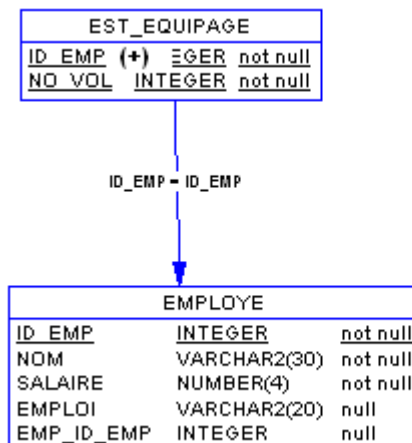
Autrement dit on affichera également `SPIROU` bien que ce dernier ne soit affecté à aucun vol.



En fait, on cherche à atteindre en plus des lignes qui satisfont la condition de jointure, les lignes de la table `EMPLOYE` dont la valeur de `Id_emp` n'apparaît pas dans la table `EST_EQUIPAGE`.

Une telle requête s'écrit en utilisant l'opérateur `+` (complémentaire) .

L'opérateur `(+)` doit être situé sur la clé étrangère qui pointe sur la table client .



```

SQL> select nom, no_vol
  2  from employe e, est_equipage eq
  3  where eq.id_emp(+) = e.id_emp
  4  order by no_vol;
  
```

NOM	NO_VOL
Gaston	1
Marilyne	1
Titeuf	2
Marilyne	2
Spirou	

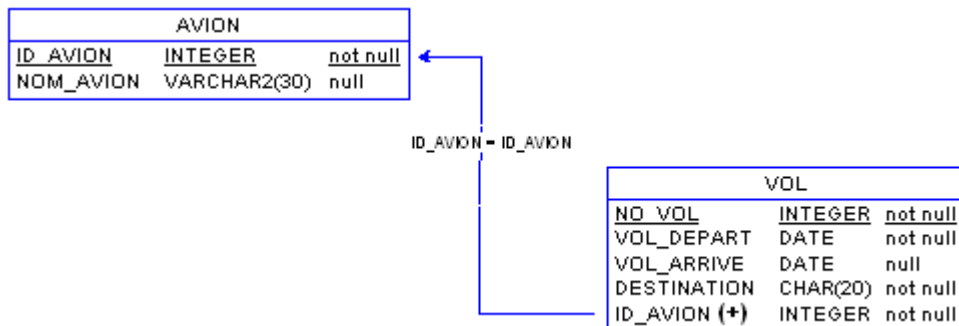
La sélection s'enrichit de Spirou qui n'est pas affecté à un vol.

Affichage de tous les avions prévus pour un vol ou non.

Pour ceux qui sont prévus pour un vol afficher le numéro de vol et la destination :







```
SQL> select nom_avion, no_vol, destination
2   from vol v, avion a
3   where v.id_avion(+) = a.id_avion
4   order by no_vol ;
```

NOM_AVION	NO_VOL	DESTINATION
Caravelle	1	Tahiti
Caravelle	2	Marquises
Boeing	3	Tokyo
Planeur		
A_Caravelle_2		

Les avions grisés n'apparaîtraient pas dans une jointure classique.

### 8.3 Ecriture et mise en forme

Le langage SQL permet la mise en forme de certaines requêtes.

#### 8.3.1 Afficher un titre de colonne

Il est possible d'afficher un libellé de son choix en titre des colonnes sélectionnées.

Dans ce cas il faudra utiliser le titre de colonne affiché dans la clause ORDER BY.

```
SQL> select id_avion "N° d'avion", nom_avion as Nom
2   from avion
3   order by "N° d'avion" desc ;
```

N° d'avion	NOM
4	A_Caravelle_2
3	Planeur
2	Bo'ng
1	Caravelle



```

SQL> select id_avion "N° d'avion", nom_avion Avion
2   from avion
3   order by "N° d'avion" desc ;

N° d'avion AVION
-----
4 A_Caravelle_2
3 Planeur
2 Bo'ng
1 Caravelle

```

### 8.3.2 Les opérateurs

Oracle possède un ensemble d'opérateurs permettant la manipulation de colonnes et de variables.

- ◆ || concaténation de chaînes de caractères
- ◆ + ajout d'un nombre à une date et heure
- ◆ - retrait d'un nombre à une date et heure
- ◆ \* multiplier
- ◆ / diviser

```

SQL> select AVG(salaire)
2   from employe ;

AVG(SALAIRE)
-----
1875

```

Vérification de la Moyenne des salaires de la société calculée précédemment.

```

SQL> select sum(salaire)/count(salaire)
2   from employe ;

SUM(SALAIRE)/COUNT(SALAIRE)
-----
1875

```

### 8.3.3 Afficher un libellé dans une requête

Il est possible d'afficher un libellé intégré à une requête dans une ligne :

```

SQL> select nom_avion || ' : Destination ==> ' || destination Iles
2   from avion a, vol v
3   where v.id_avion = a.id_avion

```



```

4 order by destination ;
-----
ILES
-----
Caravelle : Destination ==> Marquises
Caravelle : Destination ==> Tahiti
Boeing : Destination ==> Tokyo

```

## 8.4 Les fonctions

Les requêtes avec conditions de groupement s'emploient généralement avec une fonction prédéfinie.

ORACLE offre un vaste ensemble de fonctions utilisables à l'intérieur de la clause SELECT (minimum, maximum, moyenne, ...).

Beaucoup de ces fonctions ne sont pas compatibles avec d'autres SGBDR. Le problème de la migration se pose totalement et il faut se reporter au manuel ORACLE pour vérifier le respect de la norme ANSI.

### 8.4.1 Les fonctions d'agrégat

- ◆ SUM(col1,col2)                    somme des colonnes : col1 + col2
- ◆ COUNT(\*)                        nombre de lignes satisfaisant la requête
- ◆ COUNT(colonne)                nombre de lignes contenant une valeur dans la colonne
- ◆ MAX (colonne)                 maximum des valeurs de la colonne
- ◆ MIN (colonne)                minimum des valeurs de la colonne
- ◆ AVG(colonne)                 moyenne des valeurs de la colonne ; ignore les valeurs null

Ces fonctions sont utilisées pour effectuer des calculs sur un ensemble de valeurs (de lignes) d'une même colonne, dans une table.

```

SQL> select max(salaire)
2 from employe ;

MAX(SALAIRE)
-----
          2000

```

- Afficher le plus haut salaire des employés, cela ne nous dit pas combien d'employés ont ce salaire.

```

SQL> select avg(salaire) "Moyenne des salaires"
2 from employe ;

Moyenne des salaires

```



```

-----
|
|
|           1875
|
|
|

```

- Moyenne des salaires de la société, c'est la moyenne des valeurs de toutes les lignes de la table EMPLOYE pour la colonne SALAIRE.

### 8.4.2 Les fonctions numériques

- ◆ ABS (n) valeur absolue
- ◆ CEIL (n) plus petit entier égale ou supérieur à n
- ◆ POWER (m,n ) m élevé à la puissance n
- ◆ FLOOR (n) plus grand entier égal ou inférieur à n
- ◆ MOD (m,n) reste de la division de m par n (m modulo n)
- ◆ ROUND (n[,m]) n arrondi à 10 (-m) ; par défaut m = 0
- ◆ SQRT (n) racine carré de n ; retourne la valeur null si n est strictement négatif
- ◆ TRUNC (n[,m]) n tronqué à 10 (-m) ; par défaut m = 0

### 8.4.3 Les fonctions de chaînes de caractères

- ◆ SUBSTR (chaîne, a, b) = retourne une sous chaîne de a octets depuis la position b
- ◆ LTRIM(chaîne, car) = supprime les caractères à l'extrémité gauche de la chaîne 'chaîne' tant qu'ils appartiennent à l'ensemble de caractères 'car'.
- ◆ RTRIM(chaîne, car) = idem que LTRIM mais les caractères sont supprimés à droite de la chaîne.
- ◆ TRANSLATE(chaîne, car1, car2) = car1 et car2 sont des chaînes de caractères considérées comme des ensembles de caractères. La fonction TRANSLATE remplace chaque caractère de la chaîne 'chaîne' présent dans l'ensemble de caractères car1 par le caractère correspondant de la même position de l'ensemble car2.
- ◆ REPLACE(chaîne, car [, chaîne]) = permet de remplacer un ensemble de caractères 'car' par ceux de [chaîne]. Si [chaîne] est vide, les caractères 'car' sont supprimés de 'chaîne'.
- ◆ UPPER (chaîne) = converti la chaîne en majuscules
- ◆ LOWER (chaîne) = converti la chaîne en minuscules
- ◆ LENGTH (chaîne) = renvoie la longueur de la chaîne
- ◆ TO\_NUMBER = converti une chaîne de caractères en nombre
- ◆ TO\_CHAR(nombre, format) = converti un nombre en chaîne de caractères en fonction du format.



Format :

9	Représente un chiffre (non représenté si non significatif)
0	Représente un chiffre (non représenté si non significatif)
.	Point décimal apparent
V	Définit la position du point décimal non apparent
,	Une virgule apparaît à l'emplacement
\$	Un \$ précédera le premier chiffre significatif
B	Les zéros sont remplacés par des blancs
E	Le nombre est représenté avec un exposant
MI	Le signe négatif est représenté à droite
PR	Le signe négatif est placé entre <>
S	Affiche le signe '+' si la valeur est positive et le signe '-' si elle est négative
RN	Affiche la valeur en signe romain (majuscule)
rn	Affiche la valeur en signe romain (minuscule)

- ♦ `TO_CHAR(date, format)` = converti une date en chaîne de caractères en fonction du format.

Format :

SCC	Siècle (avec signe)
CC	Siècle
SY, YYY	Année (avec signe et virgule)
Y,YYY	Année (avec virgule)
YYYY	Année sur 4 chiffres
YYY	Les 3 derniers chiffres de l'année



YY	Les 2 derniers chiffres de l'année
Y	Le dernier chiffre de l'année
Q	Numéro du trimestre dans l'année
WW	Numéro de semaine dans l'année
W	Numéro de semaine dans le mois
MM	Numéro du mois
DDD	Numéro du jour dans l'année
DD	Numéro du jour dans le mois
D	Numéro du jour dans la semaine

HH	Heure sur 12 heures
HH24	Heure sur 24 heures
MI	Minutes
SS	Secondes
J	Jour du calendrier julien
YEAR	Année en toute lettres
MONTH	Nom du mois
MON	Nom du mois abrégé sur les 3 premières lettres
DAY	Nom du jour
DY	Nom du jour abrégé sur les 3 premières lettres
AM	Indication AM
PM	Indication PM
BC	Indication BC
AD	Indication AD



TH	Ajout du suffixe ordinal ST, ND, RD, TH au nombre considéré
SP	Ecriture en toutes lettres du nombre considéré
RR	Deux derniers chiffres de l'année en cours

```
SQL> col Depart format A20
SQL> select to_char(vol_depart, 'DD/MM/YYYY HH24:MI:SS') Depart,
           destination, no_vol
   2  from vol
   3  order by no_vol ;
```

DEPART	DESTINATION	NO_VOL
04/09/2004 16:19:53	Tahiti	1
09/09/2004 16:19:53	Marquises	2
30/09/2004 16:19:53	Tokyo	3

#### 8.4.4 Les fonctions de gestion du temps

- ◆ SYSDATE = retourne la date et l'heure du système d'exploitation
- ◆ NEW\_TIME(d,a,b) = transforme la date et l'heure 'd' au méridien 'a' en une date et heure au méridien 'b'.
- ◆ ROUND(date[,précision]) = arrondit la date à la précision spécifiée. La précision est spécifiée en utilisant un des masques de mise en forme de la date. Par défaut la précision est le jour.
- ◆ TRUNC(date[,précision]) = Tronque la date à la précision spécifiée.
- ◆ TO\_DATE (chaîne, format) = retourne la chaîne au format date du système (format identique à la fonction to\_char).
- ◆ ADD\_MONTHS(date, nombre) = ajoute ou soustrait le nombre de mois à la date précisée, le résultat est une date.
- ◆ MONTHS\_BETWEEN(date1, date2) = prend comme valeur la différence date1 – date2 exprimée en nombre de mois. La partie fractionnaire du résultat est calculée en considérant chaque jour comme égal à 1/31 ème jour.
- ◆ LAST\_DAY(date) = prend comme valeur la date du dernier jour du mois contenu dans (date)
- ◆ NEXT\_DAY(date, nom\_du\_jour) = prend comme valeur la date du prochain jour de la semaine spécifié par nom\_du\_jour.
- ◆ EXTRACT (sur une date ou un timestamp) = retourne la valeur d'une composante (année, mois, ...) d'une date ou d'un timestamp



```

EXTRACT(
    YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
    TIMEZONE_HOUR | TIMEZONE_MINUTE | TIMEZONE_REGION |
    TIMEZONE_ABBR FROM expression)

```

*expression* doit être de type DATE ou d'un des types TIMESTAMP

```

SQL> select to_char(sysdate, 'DAY DD MONTH YYYY') "Date du Jour"
       2 from dual ;

Date du Jour
-----
SAMEDI    11 SEPTEMBRE 2004

```

Utilisation des fonctions date pour l'insertion de lignes dans la table VOL.

```

INSERT INTO VOL VALUES
(1,sysdate,sysdate+1,'Tahiti',1 );

INSERT INTO VOL VALUES
(2,NEXT_DAY(sysdate,'JEUDI'),NEXT_DAY(sysdate,'VENDREDI'),'Marquises',1 );

INSERT INTO VOL VALUES
(3,LAST_DAY(sysdate),NULL , 'Tokyo',2 );

```

```

SQL> select * from vol;

   NO_VOL VOL_DEPA VOL_ARRI DESTINATION          ID_AVION
-----
    1 04/09/04 05/09/04 Tahiti                      1
    2 09/09/04 10/09/04 Marquises                   1
    3 30/09/04          Tokyo                      2

```

```

ALTER SESSION SET NLS_DATE_FORMAT = 'DAY MONTH DD, YYYY: HH:MIAM';

```

- Modifie le format date de la session courrante en « MONDAY JUNE 26, 2037: 10:30PM »





### 8.4.5 Autres fonctions

- ◆ `USER` = nom de l'utilisateur courant
- ◆ `NVL (expr1,expr2)` = retourne la valeur `expr2` si `expr1` est null ; sinon retourne `expr1`
- ◆ `DECODE (expr, val1,result1,(val2,result2,) ... [default])` = si `expr` égale une des valeurs, alors le résultat correspondant est retourné, sinon le résultat prend la valeur par défaut
- ◆ `ASCII ('chaîne')` = permet d'obtenir le code ASCII ou EBCDIC du premier caractère de la chaîne.
- ◆ `CHR(n)`= permet d'obtenir le caractère dont le code ASCII ou EBCDIC est égal à `n`.
- ◆ `COALESCE(expression [, ...])` = est une généralisation de la fonction `NVL`Retourne la première expression non `NULL` de la liste et remplace les valeurs `NULL` par les valeurs ou le contenu des colonnes situées entre parenthèses, dans l'ordre demandé.
- ◆ `NVL2(expression1,expression2,expression3)` = variante de la fonction `NVL`Retourne `expression2` si `expression1` est non `NULL` et `expression3` si `expression1` est `NULL`.
- ◆ `Rpad(colonne, n, expression1)` = Affiche le contenu de la colonne sur `n` caractères, et fait suivre le contenu de la colonne par `expression1` autant de fois que nécessaire pour remplir la colonne affichée.
- ◆ `NULLIF(expression1,expression2)` = `NULLIF` retourne `NULL` si deux expression sont égalesRetourne `NULL` SI `expression1` est égal à `expression2` OU retourne `expression1` SINON
- ◆ `CURSOR` = Une expression `CURSOR` retourne un curseur imbriqué, équivalent au `REF CURSOR` en PL/SQL  
Peut uniquement être utilisé dans une requête `SELECT` ou en paramètre `REF CURSOR` d'une fonction  
En SQL mais aussi en PL/SQL  
`CURSOR(sous_requête)`

```
SQL> select user, ascii('Charly')
       2 from dual ;

USER                                ASCII('CHARLY')
-----
CHARLY                                67
```

```
SQL> select no_vol, EXTRACT(YEAR FROM vol_depart) "Année de Départ"
       2 from vol;

NO_VOL Année de Départ
-----
1      2004
2      2004
3      2004
```

```
SQL> select destination, vol_arrive,
       coalesce(vol_arrive, last_day(sysdate)) "ARRIVEE BIS"
```



```

2 from vol;

DESTINATION          VOL_ARRI ARRIVEE
-----
Tahiti                05/09/04 05/09/04
Marquises             10/09/04 10/09/04
Tokyo                 30/09/04

```

```

SQL> select rpad(nom, 55, '-employe') "Nouveau nom"
2 from employe
3 order by nom ;

Nouveau nom
-----
Gaston-employe-employe-employe-employe-employe-employe-employe-
Mariline-employe-employe-employe-employe-employe-employe-employ
Spirou-employe-employe-employe-employe-employe-employe-employe-
Titeuf-employe-employe-employe-employe-employe-employe-employe-

```

## 8.5 Requêtes avec regroupement

Afficher le nombre total d'employés, affectés à un vol.

```

SQL> select no_vol, count(id_emp)
2 from est_equipage
3 group by no_vol;

NO_VOL  count(id_emp)
-----
1       2
2       2

```

Vérification :

```

SQL> select * from est_equipage
2 ;

ID_EMP  NO_VOL
-----
1       1
4       1
3       2
4       2

```

- On remarque que pour le vol N°1 l'employé 1 et l'employé 4 y sont affectés, et pour le vol 2 l'employé 3 et l'employé 4 y sont affectés.
- Le regroupement se fait en utilisant des fonctions de regroupement comme COUNT.



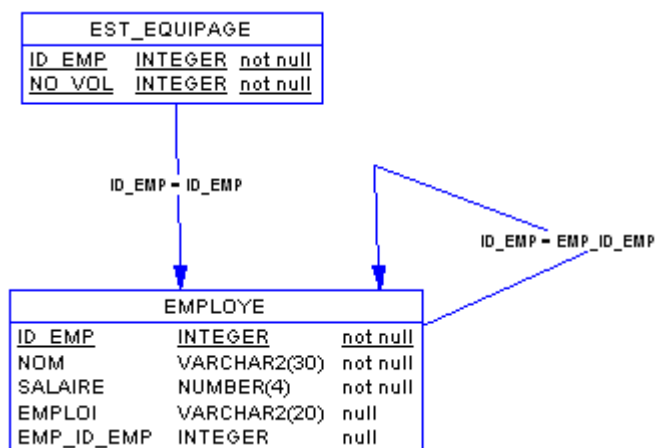
Pour rendre le regroupement de données explicite, il faut utiliser la clause `GROUP BY`.

```

SELECT  LISTE DES COLONNES DANS L'ORDRE D'AFFICHAGE
FROM    LISTE DES TABLES UTILISEES
WHERE   JOINTURE
GROUP BY REGROUPEMENT
HAVING  CONDITION
;

```

Afficher le nombre total d'employés, affectés à un vol.



```

SQL> select no_vol, count(id_emp) TOTAL
2   from est_equipage
3   group by no_vol;

```

NO_VOL	TOTAL
1	2
2	2

- Le nombre total d'employés par vol, revient à compter le nombre d' `ID_EMP` pour chaque `NO_VOL` différents.
- Le regroupement se fait sur la clé `NO_VOL`.





Dans un regroupement, il doit y avoir cohérence entre les colonnes du GROUP BY et les colonnes citées derrière le SELECT.

Afficher le nombre d'employés prévus par vol, ainsi que la destination du vol.

```
SQL> select e.no_vol, destination, count(id_emp)
  2  from vol v, est_equipage e
  3  where v.no_vol = e.no_vol
  4  group by e.no_vol
  5  ;
select e.no_vol vol, destination, count(id_emp)
      *
ERREUR Ó la ligne 1 :
ORA-00979: N'est pas une expression GROUP BY
```

- Il doit y avoir cohérence entre les colonnes derrière le SELECT et le GROUP BY.
- Ors il manque la colonne DESTINATION derrière le GROUP BY.

Affichage après correction :

```
SQL> select e.no_vol vol, destination, count(id_emp) "Nb. Employes"
  2  from vol v, est_equipage e
  3  where v.no_vol = e.no_vol
  4  group by e.no_vol, destination ;
```

VOL	DESTINATION	Nb. Employes
1	Tahiti	2
2	Marquises	2

- La clause WHERE permet d'effectuer la jointure nécessaire entre la table VOL et la table EST\_EQUIPAGE.

On aurait encore pu écrire :

```
SQL> select destination, count(id_emp) "Nb. Employes"
  2  from vol v, est_equipage e
  3  where v.no_vol = e.no_vol
  4  group by e.no_vol, destination ;
```

DESTINATION	Nb. Employes
Tahiti	2
Marquises	2



```
SQL> select destination, count(id_emp) "Nb. Employes"
  2  from vol v, est_equipage e
  3  where v.no_vol = e.no_vol
  4  group by destination ;
```

DESTINATION	Nb. Employes
Marquises	2
Tahiti	2

La condition lors d'un regroupement se fait en utilisant la clause `HAVING`

### Exemple 2

Afficher le nombre d'employés prévus pour le vol à destination de Tahiti.

```
SQL> select destination, count(id_emp) "Nb. Employes"
  2  from vol v, est_equipage e
  3  where v.no_vol = e.no_vol
  4  group by e.no_vol, destination
  5  having destination = 'Tahiti' ;
```

DESTINATION	Nb. Employes
Tahiti	2

## 8.6 Requêtes ensemblistes

Il est possible d'utiliser les opérateurs ensemblistes de l'algèbre relationnelle. Les mots clés sont `UNION`, `MINUS`, `INTERSECT`.

Il faut veiller à l'ordre des requêtes que l'on choisit de rapprocher, chacune d'elles doit correspondre à un ensemble de données, ensemble que l'on soustrait l'un de l'autre, dans un ordre établi par la requête.

Ces opérateurs sont souvent utilisés sur plusieurs tables.



Les colonnes citées derrière chaque `SELECT` doivent être de même structure (même nombre de colonne, même type de données).



### 8.6.1 *Minus*

La différence entre deux tables s'exprime par l'instruction `MINUS`. Elle permet d'afficher les lignes de la première requête qui ne sont pas comprises dans la seconde.

Afficher les avions qui ne sont pas utilisés pour un vol.

Il s'agit de la totalité des avions de la base de données MOINS les avions utilisés pour un vol.

```
SQL> select nom_avion
 2  from avion
 3  MINUS
 4  select nom_avion
 5  from vol v, avion a
 6  where v.id_avion = a.id_avion
 7  ;

NOM_AVION
-----
A_Caravelle_2
Planeur
```

### 8.6.2 *UNION*

Pour obtenir le cumul des lignes résultats de deux requêtes on utilise l'instruction `UNION`.

L'opérateur `UNION ALL` permet d'afficher les doublons.

Liste des avions de la compagnie aérienne.

C'est la liste des avions qui ne volent pas `UNION` les avions qui volent.

```
SQL> (select nom_avion
 2  from avion
 3  minus
 4  select nom_avion
 5  from vol v, avion a
 6  where v.id_avion = a.id_avion
 7  )
 8  UNION
 9  select nom_avion
10  from vol v, avion a
11  where v.id_avion = a.id_avion
12  ;

NOM_AVION
-----
A_Caravelle_2
Bo'ng
Caravelle
Planeur
```



### 8.6.3 INTERSECT

Pour obtenir les lignes résultats d'une première requête comprises également dans le résultat d'une seconde requête on pourra utiliser l'instruction `INTERSECT`.

Liste des avions qui volent, c'est l'intersection entre la liste de tous les avions et la liste des avions qui volent.

```
SQL> select nom_avion
 2  from avion
 3  intersect
 4  select nom_avion
 5  from vol v, avion a
 6  where v.id_avion = a.id_avion
 7  ;
```

NOM\_AVION

-----

Boeing  
Caravelle

### 8.7 Sous requêtes dans la clause FROM

Depuis la version 7 d'Oracle il est possible d'effectuer une sous requête dans la clause `FROM`.

La sous requête est résolue avant la requête principale.

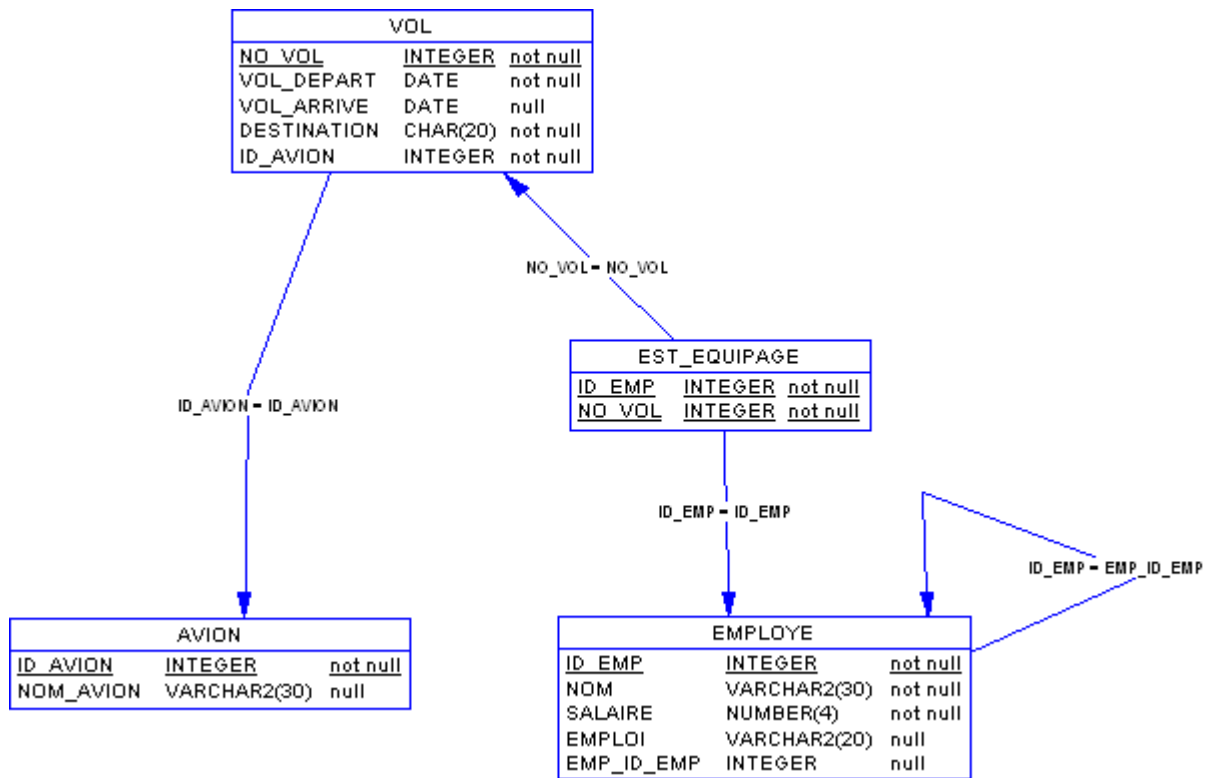
Afficher les employés travaillant dans l'avion pour le vol à destination des îles Marquises.

```
SQL> col nom format A20
SQL> col avion format A15
SQL> select nom, nom_avion Avion, destination
 2  from employe e, est_equipage eq,
 3  (select no_vol, nom_avion, destination from avion a, vol v
 4  where v.id_avion = a.id_avion and destination = 'Marquises') v_marquise
 5  where eq.no_vol = v_marquise.no_vol
 6  and eq.id_emp = e.id_emp
 7  order by nom ;
```

NOM	AVION	DESTINATION
-----	-----	-----
Marilyne	Caravelle	Marquises
Titeuf	Caravelle	Marquises

Oracle résout d'abord la sous requête de la clause `FROM` : liste des avions à destination des Marquises, puis exécute la requête principale.





## 8.8 Requêtes imbriquées

SQL offre la possibilité d’imbriquer des requêtes.  
On parle de requête principale et de sous-requête.

### 8.8.1 Opérateurs de comparaison

Le lien entre la requête principale et la sous requête peut se faire en comparant une colonne avec le résultat de la sous requête.

Ceci se fait en utilisant un opérateur arithmétique comme : <, >, =, <=, >=, <>, != .

#### Exemple

Afficher les employés qui ont un salaire au dessus de la moyenne des salaires de l’entreprise.  
Il s’agit de comparer le salaire de chaque employé avec la moyenne des salaires de l’entreprise.





La moyenne des salaires de l'entreprise est :

```
SQL> select avg(salaire)
      2 from employe ;

AVG(SALAIRE)
-----
          1875
```

Donc les employés qui ont un salaire supérieur à la moyenne des salaires sont :

```
SQL> select nom, salaire
      2 from employe
      3 where salaire > (select avg(salaire)
      4                      from employe
      5                      )
      6 order by nom ;

NOM                SALAIRE
-----
Marilyne                2000
Spirou                   2000
```

Afficher les employés qui ont le plus petit salaire

```
SQL> select nom, salaire
      2 from employe
      3 where salaire = (select min(salaire)
      4                      from employe
      5                      )
      6 order by nom;

NOM                SALAIRE
-----
Gaston                 1700
```



Si le résultat d'une sous-requête est NULL, alors la requête principale ne peut pas s'exécuter.



## 8.8.2 Opérateurs ensemblistes

Cette comparaison peut se faire également en utilisant un opérateur ensembliste.

Les opérateurs ensemblistes sont :

- ◆ ALL  
la condition est vraie si la comparaison est vraie pour chacune des valeurs retournées
- ◆ ANY  
la condition est vraie si la comparaison est vraie pour au moins une des valeurs retournées
- ◆ IN  
la condition est vraie si la comparaison est vraie pour une des valeurs retournées
- ◆ EXISTS  
Retourne le booléen vrai ou faux selon le résultat de la sous requête.



L'utilisation de IN demande des temps de réponses importants, il est fortement conseillé d'utiliser l'opérateur EXISTS.

### Exemple

Afficher les avions qui ne sont pas affectés à un vol.

#### Raisonnement :

En fait il s'agit de la liste des avions de la base de données qui ne sont pas dans la liste des avions affectés à un vol.

La liste des avions affectés à un vol correspond aux avions existants dans la table VOL .

Soit :

```
SQL> select v.id_avion, nom_avion
      2 from avion a, vol v
      3 where v.id_avion = a.id_avion ;
```

```
ID_AVION NOM_AVION
```

```
-----
      1 Caravelle
      1 Caravelle
      2 Boeing
```



La liste des avions de la base de données sont les avions de la table AVION .

Soit :

```
SQL> select id_avion, nom_avion
2  from avion
3  order by nom_avion ;
```

```
ID_AVION NOM_AVION
-----
```

```
4 A_Caravelle_2
2 Boeing
1 Caravelle
3 Planeur
```

Il en découle la requête ci-dessous :

```
SQL> select id_avion, nom_avion
2  from avion
3  where id_avion NOT IN (select v.id_avion
4                          from avion a, vol v
5                          where v.id_avion = a.id_avion
6                          )
7  order by nom_avion ;
```

```
ID_AVION NOM_AVION
-----
```

```
3 Planeur
4 A_Caravelle_2
```

Cette requête peut également s'écrire avec l'opérateur EXISTS.

Raisonnement :

Il s'agit de la liste des avions de la base de données qui n'existent pas dans la liste des avions affectés à un vol.

```
SQL> select id_avion, nom_avion
2  from avion
3  where NOT EXISTS (select v.id_avion
4                    from avion a, vol v
5                    where v.id_avion = a.id_avion
6                    and v.id_avion = avion.id_avion
7                    )
8  order by nom_avion ;
```

```
ID_AVION NOM_AVION
-----
```

```
3 Planeur
4 A_Caravelle_2
```



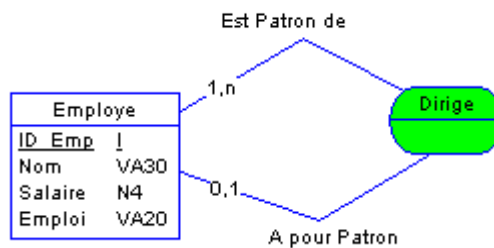
Cette requête peut également s'écrire avec l'opérateur MINUS. C'est la liste des avions moins la liste des avions affectés à un vol.

```
SQL> select nom_avion Avion
2  from avion
3  MINUS
4  select nom_avion Avion
5  from vol v, avion a
6  where v.id_avion = a.id_avion
7  ;
```

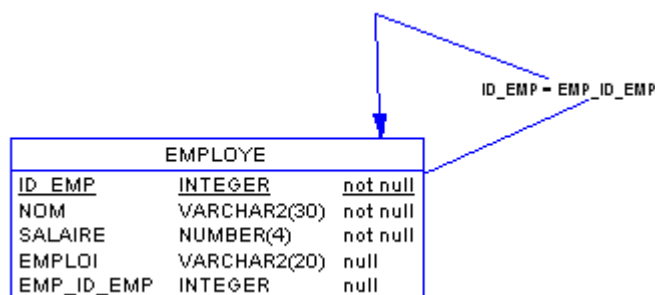
```
AVION
-----
A_Caravelle_2
Planeur
```

### 8.9 Balayer une arborescence

Le balayage d'une arborescence se fait en utilisant une relation réflexive. Dans notre base il s'agit de la relation DIRIGE.



Cette relation se traduit par l'apparition de la clé étrangère EMP\_ID\_EMP dans la table EMPLOYEE après génération du modèle logique de données.



## Exemple

Récupérer l'arborescence dont le patron a pour numéro d'employé = 1

```
SQL> select level, id_emp, nom, emp_id_emp
2     from employe
3     connect by emp_id_emp = prior id_emp
4     start with id_emp = 1
5     order by level ;
```

LEVEL	ID_EMP	NOM	EMP_ID_EMP
1	1	Gaston	
2	2	Spirou	1
2	4	Marilyne	1
3	3	Titeuf	2

Exclure seulement l'employé N° 2

```
SQL> select level, id_emp, nom, emp_id_emp
2     from employe
3     where id_emp <> 2
4     connect by emp_id_emp = prior id_emp
5     start with id_emp = 1
6     order by level ;
```

LEVEL	ID_EMP	NOM	EMP_ID_EMP
1	1	Gaston	
2	4	Marilyne	1
3	3	Titeuf	2

Exclure seulement le sous-arbre commençant par l'employé N° 2

```
SQL> select level, id_emp, nom, emp_id_emp
2     from employe
3     connect by emp_id_emp = prior id_emp
4             and emp_id_emp != 2
5     start with id_emp = 1
6     order by level ;
```

LEVEL	ID_EMP	NOM	EMP_ID_EMP
1	1	Gaston	
2	2	Spirou	1
2	4	Marilyne	1



## 9 Les jointures ANSI

La syntaxe habituelle d'écriture des jointures dans Oracle, comme nous l'avons vu précédemment n'est pas ANSI.

A partir de la version 9i, Oracle propose un ensemble d'opérateurs explicites pour réaliser la jointure de deux tables.

La syntaxe SQL 1999 n'apporte aucune amélioration en termes de performances.

Elle a été introduite par un souci de conformité avec les standards ANSI/ISO.

### 1.1 Jointures simples

L'opérateur `JOIN ON` effectue la jointure entre deux tables en se servant des conditions spécifiées respectant la syntaxe suivante :

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
    [JOIN nom_table2 ON
      (nom_table1.nom_colonne = nom_table2.nom_colonne)
;

```

- `JOIN` : indique qu'une jointure est effectuée

```
-- jointure simple
SQL> select destination, nom_avion
2   from avion a JOIN vol v
3   ON  v.id_avion = a.id_avion
4   order by destination ;

```

DESTINATION	NOM_AVION
Marquises	Caravelle
Tahiti	Caravelle
Tokyo	Bo'ng



## 2.1 Jointure avec conditions

### 9.1.1 L'opérateur JOIN ON

L'opérateur JOIN ON effectue une jointure entre deux tables respectant la syntaxe suivante :

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
    [JOIN nom_table2 ON
    (nom_table1.nom_colonne = nom_table2.nom_colonne)
    [AND | OR} Expression]]
;
```

- JOIN : indique qu'une jointure est effectuée
  - ON condition : spécifie la condition de jointure (première syntaxe)  
Remplace la condition de la clause WHERE dans la syntaxe habituelle

```
SQL> select destination, nom_avion
2   from avion a JOIN vol v
3   ON  v.id_avion = a.id_avion
4   and destination = 'Tahiti';
```

DESTINATION	NOM_AVION
Tahiti	Caravelle

### 9.1.2 L'opérateur JOIN USING

L'opérateur JOIN USING effectue une jointure entre deux tables en se servant des colonnes spécifiées respectant la syntaxe suivante :

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
    [JOIN nom_table2 USING (Nom_colonne1 [, ...] ) ]
;
```

- USING (colonne[,...]) : spécifie la condition de jointure (deuxième syntaxe) uniquement utilisable pour des équi-jointures sur des colonnes portant le même nom dans les deux tables , il indique les colonnes à utiliser pour la jointure.

Avec cette syntaxe, il est interdit de qualifier les colonnes concernées par un nom ou un alias de table.



```
SQL> select destination, nom_avion
  2  from avion
  3  JOIN vol
  4  USING (id_avion) ;
```

DESTINATION	NOM_AVION
Tahiti	Caravelle
Marquises	Caravelle
Tokyo	Bo'ng

### 9.1.3 L'opérateur NATURAL JOIN

L'opérateur NATURAL JOIN effectue la jointure entre deux tables en se servant des colonnes des deux tables qui portent le même nom.

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
  [ NATURAL JOIN nom_table2 ]
;
```

- NATURAL : indique qu'une jointure « naturelle » est effectuée, il effectue une équi-jointure sur toutes les colonnes des deux tables qui portent le même nom

```
SQL> select destination, nom_avion
  2  from avion
  3  NATURAL JOIN vol ;
```

DESTINATION	NOM_AVION
Tahiti	Caravelle
Marquises	Caravelle
Tokyo	Boeing





### 9.1.4 Produit cartésien

L'opérateur `CROSS JOIN` est un produit cartésien, il donne le même résultat qu'une requête sans condition.

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
    [ CROSS JOIN nom_table2
      CROSS JOIN nom_table3 [, ...] ]
;
```

- `CROSS JOIN` : indique qu'un produit cartésien est effectué
- En cas de jointure externe, s'il existe d'autres conditions sur les tables de la jointure externe, ces dernières doivent apparaître dans la clause `ON` ou dans la clause `WHERE` selon la table concernée :  
Dans la clause `ON` si la condition porte sur la table pour laquelle des `NULL` sont générés  
Dans la clause `WHERE` si la condition porte sur la table pour laquelle toutes les lignes sont conservées

En cas d'erreur, une jointure simple est réalisée, ou la condition n'est pas prise en compte.

```
SQL> select destination, nom_avion
2   from avion a
3   CROSS JOIN vol v
4   where v.id_avion = a.id_avion
5   ;
```

DESTINATION	NOM_AVION
Tahiti	Caravelle
Marquises	Caravelle
Tokyo	Boeing

### 9.1.5 Jointure externe

L'opérateur `OUTER JOIN` effectue une jointure externe entre deux tables en se servant des conditions spécifiées.

```
Select [ALL | DISTINCT] {*, [expression1 [AS] ALIAS1 [, ...]}
From nom_table1
    [ {LEFT | RIGHT | FULL} OUTER JOIN nom_table2 ON
      (Nom_table1.Nom_colonne = Nom_table2.Nom_colonne) ]
;
```



- LEFT | RIGHT = indique que la table de gauche | droite est dominante, celle dont on affiche toutes les lignes.
- FULL = cette option est l'union des deux requêtes : LEFT OUTER JOIN et RIGHT OUTER JOIN.

```
SQL> select destination, nom_avion
2   from avion a
3  LEFT OUTER JOIN vol v
4  ON v.id_avion = a.id_avion ;
```

DESTINATION	NOM_AVION
Tahiti	Caravelle
Marquises	Caravelle
Tokyo	Bo'ng
	A_Caravelle_2
	Planeur

Par exemple nous allons modifier de la table VOL, en désactivant des contraintes sur la colonne ID\_AVION, pour insérer une ligne dans la table VOL correspondant à un vol sans avion.

```
SQL> alter table vol disable constraint SYS_C001532;
Table modifiée.
```

```
SQL> alter table vol disable constraint FK_VOL_UTILISE_AVION;
Table modifiée.
```

```
SQL> select table_name, constraint_name, constraint_type, status
2   from user_constraints
3  order by table_name ;
```

TABLE_NAME	CONSTRAINT_NAME	C
STATUS		
-----	-----	-----
AVION	SYS_C001527	C
ENABLED		
AVION	PK_AVION	P
ENABLED		
AVION_2	SYS_C001542	C
ENABLED		
EMPLOYE	SYS_C001523	C
ENABLED		
EMPLOYE	SYS_C001524	C
ENABLED		
EMPLOYE	SYS_C001525	C
ENABLED		
EMPLOYE	PK_EMPLOYE	P
ENABLED		



```

EMPLOYE          FK_EMPLOYE_A_POUR_PA_EMPLOYE  R
ENABLED
EMPLOYE          SALAIRE_CC          C
ENABLED
EST_EQUIPAGE     SYS_C001534          C
ENABLED
EST_EQUIPAGE     SYS_C001535          C
ENABLED
EST_EQUIPAGE     PK_EST_EQUIPAGE     P
ENABLED
EST_EQUIPAGE     FK_EST_EQUI_EST_EQUIP_EMPLOYE  R
ENABLED
EST_EQUIPAGE     FK_EST_EQUI_EQUIPAGE_VOL      R
ENABLED
VOL              SYS_C001529          C
ENABLED
VOL              SYS_C001530          C
ENABLED
VOL              SYS_C001531          C
ENABLED
VOL              SYS_C001532          C
DISABLED
VOL              PK_VOL          P
ENABLED
VOL              FK_VOL_UTILISE_AVION      R
DISABLED

```

20 ligne(s) sélectionné(s).

```

SQL> insert into vol values
      2 (10, sysdate, NULL, 'Paris', NULL);

```

1 ligne créée.

Exemples de jointure externe à droite et complète (à gauche et à droite).

```

SQL> select destination, nom_avion
      2 from avion a
      3 RIGHT OUTER JOIN vol v
      4 ON v.id_avion = a.id_avion ;

```

DESTINATION	NOM_AVION
Marquises	Caravelle
Tahiti	Caravelle
Tokyo	Bo'ng
Paris	

```

SQL> select destination, nom_avion

```



```
2 from avion a
3 FULL OUTER JOIN vol v
4 ON v.id_avion = a.id_avion ;
```

DESTINATION	NOM_AVION
Tahiti	Caravelle
Marquises	Caravelle
Tokyo	Bo'ng
	A_Caravelle_2
	Planeur
Paris	

6 ligne(s) sélectionné(s).



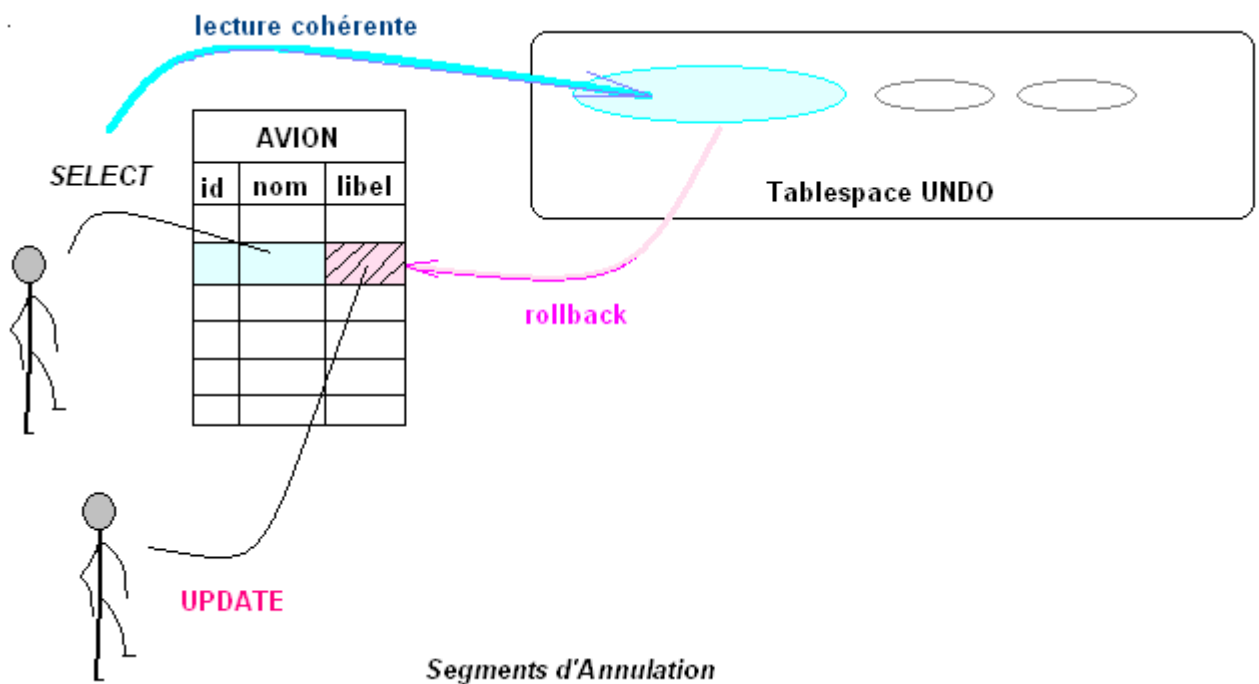
## 10 Transactions et accès concurrents

La cohérence des données repose sur le principe des transactions et des accès concurrents. Une transaction correspond à un ensemble de commandes SQL que l'on appellera actions élémentaires. Cet ensemble forme un tout qui sera entièrement validé (mise à jour définitive de la base) ou pas du tout. ORACLE offre également un mécanisme de gestion des accès concurrents. Ce mécanisme repose sur la technique du verrouillage des données. Ce verrouillage peut être implicite (par ORACLE) ou explicite (par l'utilisateur).

Principe général :

ORACLE exécute une commande qui appartient à une transaction.  
ORACLE valide une transaction dans sa globalité ou pas du tout.

La lecture cohérente garantie par Oracle est la possibilité de lire des données pendant la mise à jour de celles-ci tout en étant assuré que la version des données lues est la même.



Soit la transaction constituée des deux commandes :

```
INSERT INTO ligne_com VALUES (10,1,5,40);  
UPDATE article SET qtestock=qtestock - 40 WHERE Id_article=5;
```

La première commande insère une ligne de commande dans la table ligne\_com (la commande numéro 10 concerne 40 articles numéro 5).

La seconde commande met à jour la quantité en stock de l'article 5 d'après la quantité commandée.

Ces deux commandes doivent être exécutées et validées toutes les deux. Si, pour une raison quelconque (panne, condition fausse, ...) l'une des commandes n'a pu être traitée, ORACLE doit annuler l'autre. Lorsque les deux commandes sont exécutées et deviennent effectives, la transaction est valide. Dans le cas contraire, elle est annulée.

La base revient dans l'état qu'elle avait avant la transaction.

L'exécution d'une commande (opération élémentaire) dépend de :

- ◆ syntaxe correcte,
- ◆ respect des contraintes,
- ◆ accessibilité physique ou logique des données (réseau, droits, ...)

Pour rendre définitive l'exécution des commandes il faut valider la transaction correspondante.

La validation d'une transaction est implicite ou explicite :

- ◆ La commande commit permet de valider l'ensemble des opérations élémentaires de la transaction en cours. La prochaine opération fera débiter une nouvelle transaction.
- ◆ La commande rollback annule l'exécution des opérations élémentaires de la transaction en cours. La prochaine opération fera débiter une nouvelle transaction.
- ◆ La fin normale d'une session (programme client ou session SQL\*PLUS) entraîne la validation implicite de la transaction courante.
- ◆ La fin anormale d'une session entraîne l'annulation de la transaction courante.
- ◆ Les commandes de définition de données (CREATE, ALTER, RENAME, DROP) sont automatiquement validées.

## 10.1 Découper une transaction

Le début d'une application ou d'une session SQL constitue automatiquement le début d'une transaction. Chaque instruction commit ou rollback marque la fin de la transaction courante et le début d'une nouvelle transaction. Une transaction correspond donc à un ensemble de commandes comprises entre deux instructions commit ou rollback.



Il est cependant possible de définir plus finement une transaction en insérant des points de repères (*savepoints*).

L'instruction `SAVEPOINT` permet de préciser les points de repères jusqu'où l'annulation de la transaction pourra porter.

On crée donc ainsi des sous transactions.

```
INSERT INTO ligne_com VALUES (10,1,5,40);
SAVEPOINT point1;
UPDATE article SET qtestock=qtestock - 40 WHERE Id_article=5;
```

A ce niveau,

- l'instruction `commit` valide les deux commandes `INSERT` et `UPDATE`,
- l'instruction `rollback` annule les deux commandes `INSERT` et `UPDATE`
- l'instruction `ROLLBACK to point1` annule la commande `UPDATE`. La prochaine instruction `commit` ou `rollback` ne portera que sur la commande `INSERT`.

## 10.2 Gestion des accès concurrents

La gestion des accès concurrents consiste à assurer la sérialisation des transactions qui accèdent simultanément aux mêmes données. Cette fonctionnalité de base d'ORACLE est basée sur les concepts d'intégrité, de concurrence, et de consistance des données

### Intégrité des données

L'intégrité des données est assurée par les différentes contraintes d'intégrité définies lors de la création de la base. Elle doit être maintenue lors de l'accès simultané aux mêmes données par plusieurs utilisateurs. La base de données doit passer d'un état cohérent à un autre état cohérent après chaque transaction.

### Concurrence des données

La concurrence des données consiste à coordonner les accès concurrents de plusieurs utilisateurs aux mêmes données (deux `SELECT` doivent pouvoir s'exécuter en parallèle).

### Consistance des données

La consistance des données repose sur la stabilité des données. Lorsqu'un utilisateur utilise des données en lecture ou en mise à jour, le système doit garantir que l'utilisateur manipule toujours les mêmes données. Autrement dit, on ne doit pas débiter un traitement sur des données dont la liste ou les valeurs sont modifiées par d'autres transactions (un `SELECT` débutant avant un insert (même validé) ne doit pas afficher le nouveau *tuple* inséré)



## 10.3 Les verrous

Pour que l'exécution simultanée de plusieurs transactions donne le même résultat qu'une exécution séquentielle, la politique mise en œuvre consiste à verrouiller momentanément les données utilisées par une transaction. Dans ORACLE, le granule de verrouillage est la ligne. Tant qu'une transaction portant sur une ou plusieurs lignes n'est pas terminée (validée ou annulée), toutes les lignes sont inaccessibles en mise à jour pour les autres transactions. On parle de verrouillage. Il peut s'agir d'un verrouillage implicite ou explicite.

### Verrouillage implicite

Toute commande insert ou update donne lieu à un verrouillage des lignes concernées tant que la transaction n'est pas terminée. Toute transaction portant sur ces mêmes lignes sera mise en attente.

### Verrouillage explicite

Dans certains cas l'utilisateur peut souhaiter contrôler lui-même les mécanismes de verrouillage.

En général, il utilise la commande :

```
|select * from vol for update
```

Tous les `VOLs` sont verrouillés mais une clause `WHERE` est possible. Le verrouillage ne porte alors que sur les lignes concernées.

Il existe différents modes de verrouillages d'une table (mode lignes partagées, équivalent au *select for update*, mode lignes exclusives, mode table partagée, mode partage exclusif, mode table exclusive).

En plus de la simple visibilité des données, on peut ainsi préciser les verrous autorisés par dessus les verrous que l'on pose. Par exemple, plusieurs *select for update* peuvent s'enchaîner (verrouillage en cascade).

Lorsque la première transaction sera terminée, le second *select for update* pose ses verrous et ainsi de suite. Par contre, un verrouillage en mode table exclusive empêche tout autre mode de verrouillage. A titre d'exemple, nous ne présenterons ici que les verrouillages standards (implicites suite à une commande insert, update, ou delete).

### Verrouillage bloquant

ORACLE détecte les verrouillages bloquant (*deadlock*). Ces verrouillages correspondent à une attente mutuelle de libération de ressources.





## Exemple

## Transaction T1

```
update article set qtestock=10 where Id_article=1; update article set qtestock=30 where Id_article=2;
```

```
update article set qtestock=20 where Id_article=2; update article set qtestock=40 where Id_article=1;
```

```
commit;
```

## Transaction T2

```
commit;
```

Si les deux transactions ne sont pas lancées « vraiment » en même temps, on ne parle pas de verrouillage bloquant. Les deux transactions s'exécutent normalement l'une à la suite de l'autre.



Dans tous les cas, après une instruction commit ou rollback :  
 Les verrous sont levés  
 Une nouvelle transaction commence à la prochaine instruction.

## 10.4 Accès concurrents en mise à jours

Si deux utilisateurs accèdent à des lignes différentes d'une table qui n'a pas fait l'objet d'un verrouillage particulier, les transactions s'effectuent normalement.

Si les deux utilisateurs accèdent aux mêmes lignes d'une table alors la transaction débutée le plus tard sera mise en attente. La validation de la première transaction « libérera » la seconde.

Les mécanismes internes de gestions des transactions et des accès concurrents sont gérés par ORACLE. Il reste à la charge du programmeur la gestion des verrous explicites et la maîtrise des verrous implicites. Les règles générales sont les suivantes :

Une transaction est constituée d'un ensemble d'opérations élémentaires (insert, update, ...),

ORACLE garantit qu'une transaction est entièrement validée ou défaite,

Toute session SQL (sous SQL\*PLUS ou depuis un programme) démarre une transaction,

Toute fin normale de session déclenche un commit,

Toute fin anormale de session déclenche un rollback,

L'unité de verrouillage sous ORACLE est la ligne,

Une commande INSERT, DELETE, ou UPDATE entraîne un verrouillage implicite des lignes concernées,

La commande SELECT FOR UPDATE permet de verrouiller explicitement les lignes concernées. Elle peut utiliser la clause WHERE pour ne pas verrouiller toute la table,

Les verrous sont levés par les commandes commit ou rollback.

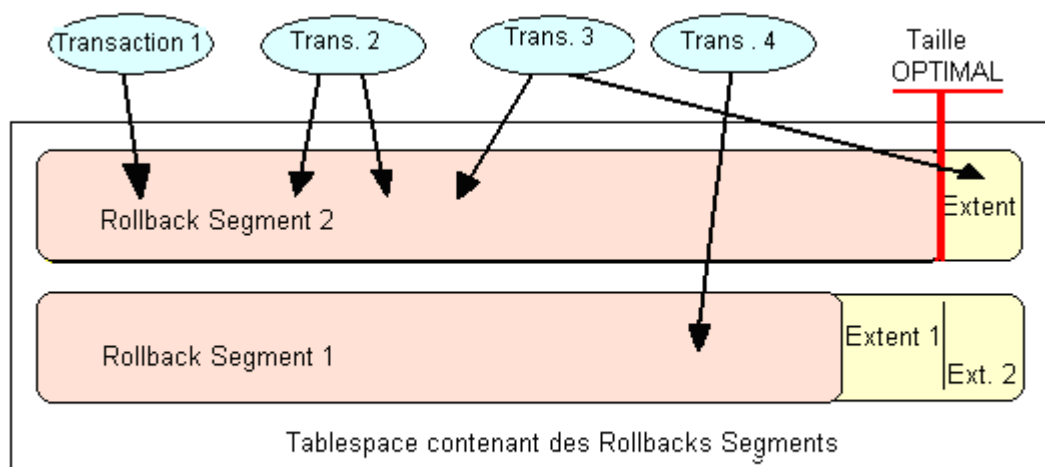


Ne jamais perdre de vue les scénarii d'activité des opérateurs afin d'éviter de mettre en place une gestion aussi fine qu'inutile de l'unité de verrouillage (ligne ?, table?). Concrètement, il faut se poser des questions de base comme « Combien d'accès concurrents sur telles données observe-t-on en moyenne ? ». Le code s'en trouvera considérablement simplifié.

## 10.5 Les rollbacks segments ou segments d'annulation

Les rollbacks segments sont des segments permettant à Oracle de stocker l'image avant les modifications effectuées durant une transaction.

C'est Oracle qui alloue les transactions aux rollback segments.



*Les Rollbacks Segments*

Lorsque la transaction se termine, elle libère le rollback segment mais les informations de rollback ne sont pas supprimées immédiatement

Ces informations peuvent encore être utiles pour une lecture cohérente

Par défaut c'est Oracle qui alloue les rollback segment aux transactions en cherchant à répartir les transactions concurrentes sur les différents rollback segment. Dans certain cas il est possible d'allouer un rollback segment à une transaction en utilisant l'ordre SQL : `SET TRANSACTION USE ROLLBACK SEGMENT.`

Lorsqu'un rollback segment est plein et que la transaction a besoin d'espace, une erreur se produit et la transaction est arrêtée. Le rollback segment grossit dans la limite de la taille du tablespace qui le contient. En cas d'erreur, il faut alors retailler le rollback segment puis relancer la transaction en lui affectant le rollback segment agrandi.

L'erreur « *snapshot to hold* » correspond à problème de lecture cohérente. Une requête (`SELECT`) dans un segment peut être écrasée par une transaction, lors de la lecture cohérente si il y a besoin de cette requête (`SELECT`) cela provoque l'erreur « *snapshot to hold* ».



## 11 Modifier les lignes de tables

La mise à jour des données d'une base se fait par l'une des commandes suivantes :

INSERT	Insertion d'une ligne
UPDATE	Modification d'une ou plusieurs lignes
DELETE	Suppression d'une ou plusieurs lignes

Les commandes de mise à jour de la base déclenchent éventuellement des triggers (cf. chapitre TRIGGERS) ou des contraintes d'intégrité. Elles n'accèdent donc pas directement aux données comme en témoigne le schéma suivant :

Nous allons présenter les points fondamentaux de la syntaxe de ces commandes (nous appuyons nos exemples sur le schéma de la base exemple précédente).

### 11.1 Insérer des lignes dans une table

#### 11.1.1 La commande INSERT

La commande INSERT permet d'insérer une ligne dans une table.

```
INSERT INTO nom_table
VALUES (liste de valeurs séparées par des virgules dans l'ordre des
colonnes créées);
```

```
INSERT INTO nom_table (liste de colonnes séparées par des virgules dans
l'ordre créés)
VALUES (liste de valeurs séparées par des virgules dans l'ordre des
colonnes citées);
```

- Les CHAR et VARCHAR doivent être saisis entre apostrophes '...'
- La valeur NULL permet de ne pas saisir un champ
- La fonction to\_date permet de traduire une date dans le format interne.

```
----- INSERT Avion -----
--
INSERT INTO Avion VALUES
(1,'Caravelle' );
INSERT INTO Avion VALUES
(2,'Boïng' );
INSERT INTO Avion VALUES
```



```
(3,'Planeur' );
insert into avion values
(4,'A_Caravelle_2');

----- INSERT Vol -----
--
INSERT INTO VOL VALUES
(1,sysdate,sysdate+1,'Tahiti',1 );
INSERT INTO VOL VALUES
(2,NEXT_DAY(sysdate,'JEUDI'),NEXT_DAY(sysdate,'VENDREDI'),'Marquises',1 );
INSERT INTO VOL VALUES
(3,LAST_DAY(sysdate),NULL ,'Tokyo',2 );
```

**Vérification :**

```
SQL> select * from avion;

   ID_AVION NOM_AVION
-----
          1 Caravelle
          2 Bo'ng
          3 Planeur
          4 A_Caravelle_2

SQL> select * from vol;

   NO_VOL VOL_DEPA VOL_ARRI DESTINATION          ID_AVION
-----
          1 04/09/04 05/09/04 Tahiti          1
          2 09/09/04 10/09/04 Marquises       1
          3 30/09/04          Tokyo          2
```

**11.1.2 Insertion à partir d'une table existante**

Nous allons créer une table AVION\_2, car pour notre exemple il faut travailler obligatoirement sur une autre table.

```
SQL> create table avion_2
2  (
3      ID_AVION      INTEGER          not null,
4      NOM_AVION     VARCHAR2(30)      null ,
5      constraint PK_AVION_2 primary key (ID_AVION),
6      DESTINATION   VARCHAR2(30)      null
7  );
```

Table créée.

```
SQL> desc avion_2
Nom                                NULL ?   Type
-----
ID_AVION                           NOT NULL NUMBER(38)
NOM_AVION                           VARCHAR2(30)
```



```
DESTINATION VARCHAR2(30)
```

```
SQL> select * from avion_2;
aucune ligne sélectionnée
```

```
SQL> insert into avion_2
2  select a.id_avion, nom_avion, destination
3  from avion a, vol v
4  where v.id_avion = a.id_avion
5  and destination = 'Marquises' ;
```

1 ligne créée.

```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises

```
SQL> insert into avion_2 (id_avion, nom_avion)
2  select id_avion, nom_avion
3  from avion
4  where id_avion > 1 ;
```

3 ligne(s) créée(s).

```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2	Bo'ng	
3	Planeur	
4	A_Caravelle_2	

## 11.2 Modifier les lignes d'une table

### 11.2.1 La commande UPDATE

La commande UPDATE permet de modifier une ou plusieurs lignes d'une table.

```
UPDATE nom_table SET liste d'affectations
WHERE conditions sur les lignes concernées;
```





Sans clause WHERE, toute la table est modifiée

```
SQL> select * from vol
2 ;

NO_VOL VOL_DEPA VOL_ARRI DESTINATION ID_AVION
-----
1 04/09/04 05/09/04 Tahiti 1
2 09/09/04 10/09/04 Marquises 1
3 30/09/04 Tokyo 2

SQL> update vol set
2 vol_arrive = to_date('01/10/2004 03:30:00', 'DD/MM/YYYY HH24:MI:SS')
3 where no_vol = 3 ;

1 ligne mise à jour.
```

Vérification :

```
SQL> col depart for A20
SQL> col arrive for A20
SQL> select to_char(vol_depart, 'DD/MM/YYYY HH24:MI:SS') Depart,
2 to_char(vol_arrive, 'DD/MM/YYYY HH24:MI:SS') Arrive,
3 destination
4 from vol
5 where no_vol = 3 ;

DEPART ARRIVE DESTINATION
-----
30/09/2004 16:19:53 01/10/2004 03:30:00 Tokyo
```

### 11.2.2 Modifications de lignes à partir d'une table existante

Dans cet exemple nous allons modifier la table AVION\_2 créée précédemment.

```
SQL> select * from vol ;

NO_VOL VOL_DEPA VOL_ARRI DESTINATION ID_AVION
-----
1 04/09/04 05/09/04 Tahiti 1
2 09/09/04 10/09/04 Marquises 1
3 30/09/04 01/10/04 Tokyo 2
```



Modification de la table :

```
SQL> update avion_2
  2   set (destination) = (select destination
  3                           from vol
  4                           where no_vol = 1)
  5   where destination is null ;
```

3 ligne(s) mise(s) à jour.

```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2	Boeing	Tahiti
3	Planeur	Tahiti
4	A_Caravelle_2	Tahiti

### 11.3 Spécifier la valeur par défaut d'une colonne

Dans un ordre `INSERT` ou `UPDATE`, il est possible d'affecter explicitement à une colonne la valeur par défaut définie sur cette colonne

En mettant le mot clé `DEFAULT` comme valeur de la colonne `NULL` est affecté si la colonne n'a pas de valeur par défaut

Lors d'un `INSERT` :

```
SQL> insert into avion_2
  2   values (5, 'Petit coucou', 'Canaries');
```

1 ligne créée.

```
SQL> insert into avion_2
  2   values (6, 'Petit coucou', default);
```

1 ligne créée.

```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2	Bo'ng	Tahiti
3	Planeur	Tahiti
4	A_Caravelle_2	Tahiti
5	Petit coucou	Canaries
6	Petit coucou	

6 ligne(s) sélectionnée(s).



Lors d'un UPDATE :

Lors d'un UPDATE :

```
SQL> update avion_2
  2 set nom_avion = default
  3 where id_avion = 5 ;
```

1 ligne mise à jour.

```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2	Bo'ng	Tahiti
3	Planeur	Tahiti
4	A_Caravelle_2	Tahiti
5		Canaries
6	Petit coucou	

6 ligne(s) sélectionné(s).

```
SQL> update avion_2
  2 set nom_avion = default
  3 where destination like '%h%';
```

3 ligne(s) mise(s) à jour.

```
SQL> select * from avion_2 ;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2		Tahiti
3		Tahiti
4		Tahiti
5		Canaries
6	Petit coucou	

## 11.4 Supprimer les lignes d'une table

### 11.4.1 La commande DELETE

La commande DELETE permet de supprimer une ou plusieurs lignes d'une table.

```
DELETE FROM nom_table
  WHERE conditions sur les lignes concernées;
```







Sans la clause WHERE toute la table est vidée.

Suppression du vol numéro 10, et vérification.

```
SQL> select * from vol ;
```

NO_VOL	VOL_DEPA	VOL_ARRI	DESTINATION	ID_AVION
1	04/09/04	05/09/04	Tahiti	1
2	09/09/04	10/09/04	Marquises	1
3	30/09/04		Tokyo	2
10	11/09/04		Paris	

```
SQL> delete from vol where no_vol = 10;
```

1 ligne supprimée.

```
SQL> select * from vol ;
```

NO_VOL	VOL_DEPA	VOL_ARRI	DESTINATION	ID_AVION
1	04/09/04	05/09/04	Tahiti	1
2	09/09/04	10/09/04	Marquises	1
3	30/09/04		Tokyo	2

Supprimer toutes les lignes de la table AVION\_2 sans destination :.

```
SQL> select * from avion_2 ;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises
2	Bo'ng	
3	Planeur	
4	A_Caravelle_2	

```
SQL> delete from avion_2
2 where destination is null ;
```

3 ligne(s) supprimé(s).



```
SQL> select * from avion_2;
```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises

### 11.4.2 Vider une table

Le vidage d'une table supprime toutes les lignes de la table et libère l'espace utilisé.

La table et ses index sont supprimés.

Une table référencée par une clé étrangère ne peut pas être supprimée car la contrainte est vérifiée.

Le vidage d'une table se fait en utilisant la commande suivante :

```
truncate table [ shema. ] nom_table
  [ { drop | reuse } storage ]
;
```

- Si le paramètre DROP est utilisé, tous les extents sont supprimés.
- Si le paramètre REUSE est spécifié, l'espace utilisé par la table est conservée.

```
SQL> truncate table est_equipage ;
```

Table tronquée.

```
SQL> truncate table avion;
truncate table avion
*
```

```
ERREUR Ó la ligne 1 :
ORA-02266: Les clés primaires/unique de la table référencées par des clés
étrangères
```



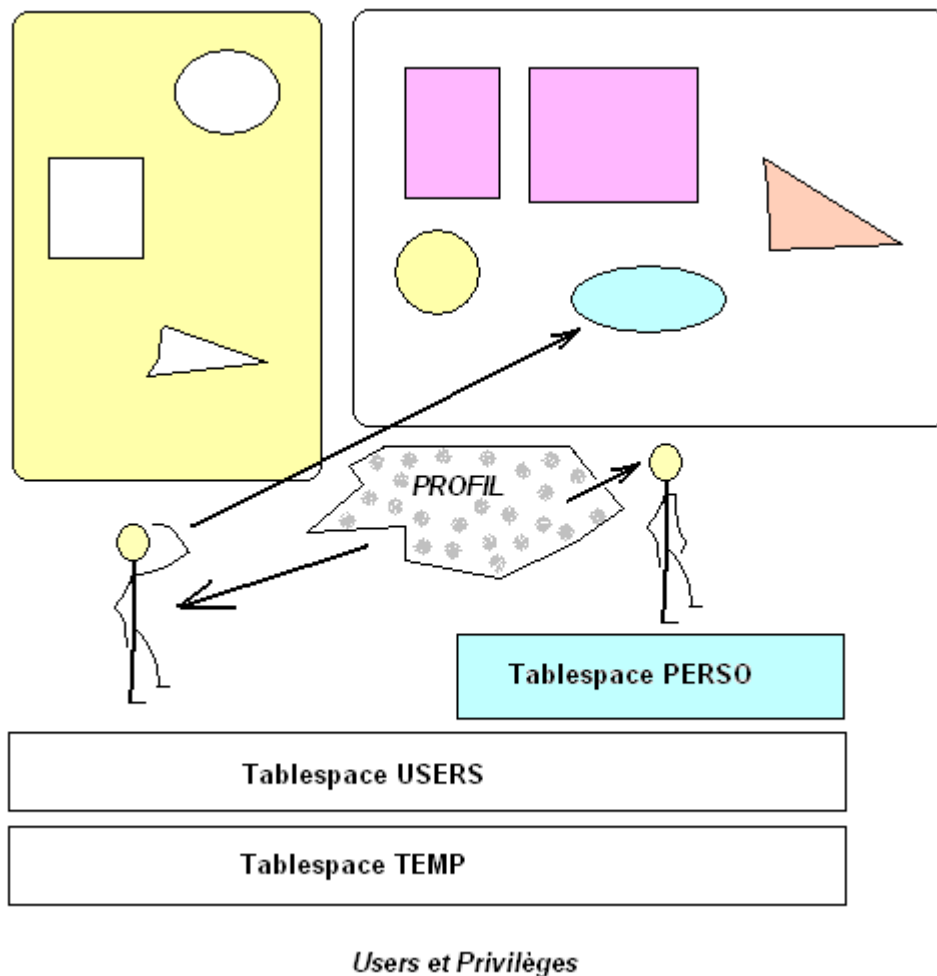
Lors du vidage d'une table, il faut inactiver les contraintes clé étrangère si nécessaire.  
Ne pas oublier de les réactiver après !



## 12 gestion de la confidentialité

Oracle permet à plusieurs utilisateurs de travailler en toute sécurité sur la base de données.

Chaque objet peut être définie, soit comme confidentielle et accessible à un nombre restreint d'utilisateurs, soit accessible à l'ensemble des utilisateurs.



Les ordres `GRANT` et `REVOKE` du langage SQL permettent de définir les droits de chaque utilisateur sur les objets de la base de données.





Un privilège est le droit :

- D'exécuter un ordre SQL en général (par exemple, créer une table  
Privilège système
- D'accéder à un objet d'un autre utilisateur (par exemple, mettre à jour les données de la table CLIENT  
Privilège objet

Les privilèges peuvent être attribués directement aux utilisateurs ou par l'intermédiaire de rôles. Un rôle est un regroupement nommé de privilèges.

## 12.1 Gestion de la confidentialité niveau objet

Pour accéder à un objet, le propriétaire doit vous donner un des privilèges objet suivants :

Privilège	Table	Vue	Séquence	Procédure
SELECT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
INSERT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
UPDATE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
DELETE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
EXECUTE				<input checked="" type="checkbox"/>
ALTER	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
INDEX	<input checked="" type="checkbox"/>			
REFERENCES	<input checked="" type="checkbox"/>			



Les privilèges INSERT, UPDATE et REFERENCES peuvent être restreints à certaines colonnes.



```
@ Creer_TS_Tahiti

-- MAJ spfile pour prise en compte de
-- la limitation des ressources
ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;

-- SUPPRESSION DU PROFILE PRODEFI
drop profile prodefi cascade ;

-- CREATION PROFILE PRODEFI
create profile prodefi limit
    sessions_per_user 1
    cpu_per_session 6000
    cpu_per_call 6000
    idle_time 30;

-- visualiser la liste des privilèges système avec la requête si dessous.
-- Select name from system_privilege_map
-- Order by name ;

-- Suppression du role rolstage
Drop role rolstage cascade;

-- CREATION DU ROLE stage (SQL + PL)
create role rolstage;
grant connect to rolstage;
grant resource to rolstage;
grant create public synonym to rolstage;

-- CREATION USER ET ATTRIBUTION DU ROLE
Drop user charly cascade ;

-- Creation du user Charly
create user charly identified by defi
    default tablespace TOOL
    quota unlimited on DATA
    quota unlimited on INDX
    profile prodefi;

grant rolstage to charly;

connect charly/defi@tahiti
@ Creer_Tables_Tahiti
@ Rempli_Tables_Tahiti
```



## 12.2 Gestion de la confidentialité niveau system

Chaque ordre SQL a au un moins un privilège système associé.

Souvent, l'ordre SQL a un privilège système qui porte le même nom, par exemple, l'ordre `CREATE TABLE` a un privilège système associé qui s'appelle `CREATE TABLE` (donne le droit de créer une table dans son propre schéma).

Certains privilèges système reprennent le nom de l'ordre SQL avec le mot clé `ANY`, dans ce cas, le privilège système permet d'exécuter l'ordre dans n'importe quel schéma de la base, par exemple, le privilège système `CREATE ANY TABLE` donne le droit de créer une table dans n'importe quel schéma de la base.



La liste des privilèges système est accessible grâce à la vue `SYSTEM_PRIVILEGE_MAP`.

```
SQL> desc system_privilege_map
Nom                                NULL ?   Type
-----
PRIVILEGE                          NOT NULL NUMBER
NAME                                NOT NULL VARCHAR2(40)
PROPERTY                            NOT NULL NUMBER

SQL> Select name from system_privilege_map
  2 where name like '%CREATE%'
  3 Order by name ;

NAME
-----
CREATE ANY CLUSTER
CREATE ANY CONTEXT
CREATE ANY DIMENSION
CREATE ANY DIRECTORY
CREATE ANY EVALUATION CONTEXT
CREATE ANY INDEX
CREATE ANY INDEXTYPE
CREATE ANY LIBRARY
CREATE ANY OPERATOR
CREATE ANY OUTLINE
CREATE ANY PROCEDURE
CREATE ANY RULE
CREATE ANY RULE SET
CREATE ANY SECURITY PROFILE
CREATE ANY SEQUENCE
CREATE ANY SNAPSHOT
...
```



## 12.3 Les rôles

Les rôles permettent de simplifier la gestion des droits.



Un rôle peut être attribué à un autre rôle.  
Un utilisateur peut avoir plusieurs rôles.  
Un rôle n'appartient à personne.

La mise en œuvre s'effectue en trois étapes :

Création du rôle

Attribution des privilèges (système et objet) au rôle

Attribution du rôle aux utilisateurs



Lorsque le rôle est enlevé, les utilisateurs connectés peuvent toujours exercer les privilèges associés à ce rôle jusqu'à la fin de la session, ou la désactivation du rôle.

Un rôle attribué à un utilisateur (directement ou via un autre rôle) est par défaut automatiquement activé lors de la connexion de l'utilisateur.

Si l'utilisateur est connecté au moment de l'attribution, l'activation immédiate n'est pas automatique :

Mais l'utilisateur peut l'activer par l'ordre SQL `SET ROLE`

```
SET ROLE
{ nom_rôle [ IDENTIFIED BY mot_de_passe ] [,...]
  | ALL [ EXCEPT nom_rôle [,...] ] | NONE }
;
```

- IDENTIFIED BY = donne le mot de passe qui permet d'activer le rôle.
- ALL = active tous les rôle attribués à l'utilisateur.
- La clause EXCEPT permet d'en enlever certains
- NONE = désactive tous les rôles.

```
-- L'utilisateur VDEP active le rôle MAILING
SET ROLE rolstage;
```



Utiliser plusieurs rôles sans qu'ils soient tous actifs présente deux intérêts :

- ◆ Le paramètre `MAX_ENABLED_ROLES` (20 par défaut) limite le nombre de rôles actifs simultanément pour un utilisateur. Si un utilisateur utilise un nombre de rôles supérieur à cette limite, il est possible d'en désactiver certains pour en activer d'autres.
- ◆ Des rôles protégés par un mot de passe peuvent être attribués à des utilisateurs, mais inactifs par défaut et sans donner le mot de passe à l'utilisateur. C'est l'applicatif qui active le rôle en fournissant le mot de passe de façon transparente. Par exemple l'application se connecte et lorsque l'utilisateur se signe le rôle devient actif.

```
ALTER USER charly DEFAULT ROLE rolstage;  
ALTER USER clo DEFAULT ROLE ALL EXCEPT rolstage;
```

Oracle propose une douzaine de rôles prédéfinis dont :

- ◆ `Connect` : permet la création des principaux objets d'un schéma (table, vues, ...)
- ◆ `Ressource` : permet la création des principaux objets d'un schéma (table, vues, ...)
- ◆ `Db` : donne tous les privilèges système avec la clause `WITH ADMIN OPTION`
- ◆ `Exp_full_database` : permet l'export complet de la base
- ◆ `Imp_full_database` : permet l'import complet de la base
- ◆ `Select_catalog_role` : permet de lire le dictionnaire de données (accéder aux vues `DBA_` et `V$` )
- ◆ `Execute_catalog_role` : permet d'exécuter les packages du dictionnaire de données
- ◆ `Delete_catalog_role` : permet de supprimer dans les tables du dictionnaire de données.






## 13 Notion de schéma

Un schéma est le regroupement des objets d'un utilisateur dans une même unité logique.

Il permet de construire l'ensemble des structures d'une application en une seule opération.

Le contrôle des dépendances entre les objets est réalisé à la fin de la création de tous les objets.



Les objets appartenant à un utilisateur sont préfixés par le nom de l'utilisateur.  
 ce sont des objets de schéma !

### 13.1 Création d'un schéma

La commande `CREATE SCHEMA` permet de créer un schéma :

```
CREATE SCHEMA AUTHORIZATION NOM_SEQUENCE
CREATE TABLE NOM_TABLE
CREATE VIEW NOM_VUE
GRANT LISTE_PRIVILEGES ON {NOM_TABLE|NOM_VUE} TO `USER|ROLE`
;
```



Le nom du schéma ne peut être que celui du propriétaire des objets créés.

```
Create scheme authorization cours1
Create table emp(no number(7) not null,
                Nom varchar2(25) not null,
                Dt_entrée dat not null,
                No_service number(2),
                Constraint emp_no_pk primary key (no))
Tablespace user_data
Storage (initial 10K
```



```

        Next 10K
        Pctincrease 0)

Grant select, update on emp to cours2

Create table service (no number(2) not null,
                    Nom varchar2(15) not null)
Tablespace user_data
Storage (initial 10K
        Next 10K
        Pctincrease 0)

Grant select on service to cours 2
/

```

## 13.2 Intérêt d'un schéma

Il facilite la gestion des objets utilisateur : si une seule opération échoue lors de la création du schéma (création d'une table), il y a un « rollback » général et toutes les opérations sont annulées, aucun objet n'est créé.

Il facilite l'administration des statistiques sur les objets, au lieu d'exécuter une commande SQL pour chaque objet, on peut le faire pour tous les objets appartenant au schéma.



Tout est validé ou rien ne l'est. Technique difficile avec un nombre important de tables, de vues et de packages.

```

| Execute dbms_utility.analyze_schema ('nom_schema', 'compute|estimate|delete')
| ;

```

```

| Execute dbms_utility.analyze_schema ('cours1', 'compute')
| ;

```

## 13.3 Modifier un élément de schéma

Il existe deux méthodes possibles :

Supprimer et recréer l'objet par les commandes `DROP` et `CREATE`, dans ce cas il faudra créer à nouveau les privilèges.

Renommer l'objet par la commande `RENAME TO`, l'objet conserve alors ses privilèges d'origine.





On ne peut renommer ni un traitement catalogué, ni un synonyme public, ni un cluster. Dans tous les cas les objets dépendants seront invalides et seront recompilés.



## 14 Les objets de schema

### 14.1 Les vues

Les vues offrent :

#### Sécurité

L'administrateur permet d'accéder seulement à des vues dont il peut également administrer l'accès (insert, update, delete, select).

#### Masquer la complexité

L'utilisateur risque moins de se tromper sur des requêtes complexes. La complexité de la requête se trouve dans le texte de création de la vue.

#### Simplifier la formulation

Un simple *select \* from nom\_vue* peut être en fait une requête mettant en oeuvre de nombreuses jointures.

#### Sauvegarde indirecte de requêtes complexes

On est sûr que le code d'une requête stockée dans une vue est le même pour tous (principe d'encapsulation)

#### Gestion des profils utilisateurs

La perception logique des données n'est pas la même pour tous les utilisateurs. Les vues facilitent la gestion de ces différences. Chaque profil d'utilisateur utilisera la vue qui le concerne.

#### 14.1.1 Créer une vue

Nous allons présenter le concept de vue à travers une suite progressive d'exemples. Une synthèse sur l'intérêt des vues est proposée à la fin de ce chapitre.

Une vue est une perception logique d'une ou plusieurs tables (ou vues) définie à partir d'une requête.

Création d'une vue sur les avions et les vols prévus.

```
SQL> CREATE VIEW Mes_Avions
  2   AS select v.id_avion, nom_avion, destination
  3         from avion a, vol v
  4         where v.id_avion = a.id_avion
  5   ;
```

Vue créée.

L'accès aux éléments d'une vue est le même que pour ceux d'une table ...



```
SQL> select * from mes_avions;

  ID_AVION NOM_AVION                DESTINATION
-----
         1 Caravelle                Tahiti
         1 Caravelle                Marquises
         2 Boeing                    Tokyo
```

La syntaxe utilisée ici semble exprimer qu'une vue et une table sont de même nature. Il n'en est rien. Une vue ne nécessite pas d'autre stockage d'information que le texte de sa requête de création et une entrée dans le dictionnaire des vues.



Chaque fois que l'on manipule une vue, le texte effectif de la requête est reconstruit dynamiquement en consultant le dictionnaire des vues.

La vue du `USER_VIEWS` du dictionnaire de données permet d'afficher le contenu de la vue.

```
SQL> select view_name, text
       2 from user_views
       3 ;

VIEW_NAME
-----
TEXT
-----
MES_AVIONS
select v.id_avion, nom_avion, destination
       from avion a, vol v
```

Du point de vue fonctionnel, les vues supportent toutes les opérations SQL comme `INSERT`, `UPDATE`, `DELETE`, `SELECT`.

A travers la vue, c'est en fait la table à partir de laquelle la vue a été construite qui sera mise à jour (et la vue généralement aussi par conséquence).

Si il est possible de modifier une table à travers une vue, mais il existe des contraintes importantes :

```
SQL> insert into mes_avions
       2 values (11, 'Coucou', 'Perou')
       3 ;
insert into mes_avions
*
ERREUR Ó la ligne 1 :
ORA-01776: Impossible de modifier plus d'une table de base via une vue jointe
```



```
SQL> insert into mes_avions
  2      (destination)
  3 values ('Perou')
  4 ;
insert into mes_avions
*
ERREUR Ó la ligne 1 :
ORA-01400: impossible d'insÚrer NULL dans ("CHARLY"."VOL"."NO_VOL")
```

La vue n'utilise pas des opérateurs tels que : GROUP BY, DISTINCT, ORDER BY, des fonctions d'agrégat, ou des fonctions analytiques, des collections ou des requêtes imbriquées...

- ◆ La table ne contient pas de colonne de type LOB ou de type objet
- ◆ Les contraintes d'intégrité sont respectées à travers la vue
- ◆ La table ne fait pas l'objet de réplication
- ◆ Les index de la table sont de type B-Tree (pas de Cluster ou d'IOT : *Index Organised Table*)



Pour la modification, la vue doit contenir une clause :  
WITH CHECK OPTION

Les options que l'on peut associer à une vue lors de sa création sont :

WITH CHECK OPTION	Respecte les conditions de la vue en mise à jour
WITH READ ONLY	N'autorise que la lecture

La clause WITH CHECK OPTION garantit que toutes les insertions ou les mises à jour à travers la vue seront maintenant contrôlées avant d'être effectives. Le contrôle effectué correspond aux conditions précisées dans la vue.

```
SQL> create view Mes_Vols
  2 as select no_vol, vol_depart, destination
  3 from vol
  4 where vol_depart > to_date('10/09/2004', 'DD/MM/YYYY')
  5 with check option ;
```

Vue créée.

```
SQL> select * from mes_vols;

NO_VOL VOL_DEPA DESTINATION
-----
      3 30/09/04 Tokyo
```



Attention aux contraintes d'intégrités, par exemple l'insertion d'un nouveau vol à travers la vue est contrôlée ...

```
SQL> insert into mes_vols
  2 values (11,to_date('12/09/2004 20:30:00', 'DD/MM/YYYY HH24:MI:SS'), 'Perou');
insert into mes_vols
*
ERREUR Ó la ligne 1 :
ORA-01400: impossible d'insérer NULL dans ("CHARLY"."VOL"."ID_AVION")
```

### Exemple

Nous allons recréer la vue Mes\_Vols en y ajoutant la colonne ID\_AVION de la table VOL.

Nous visualiserons dans l'exemple l'insertion de lignes dans la vue MES\_VOLS et le contrôle fait par Oracle avec l'option WITH CHECK OPTION.

```
SQL> Drop view mes_vols;
Vue supprimée.

SQL> create view Mes_Vols
  2 as select no_vol, vol_depart, destination, id_avion
  3 from vol
  4   where vol_depart > to_date('10/09/2004', 'DD/MM/YYYY')
  5   with check option ;

Vue créée.

SQL> insert into mes_vols
  2 values (11,to_date('12/09/2004 20:30:00', 'DD/MM/YYYY HH24:MI:SS'),
  3         'Perou', 3);

1 ligne créée.

SQL> select * from mes_vols;

   NO_VOL VOL_DEPA DESTINATION          ID_AVION
-----
      3 30/09/04 Tokyo                    2
     11 12/09/04 Perou                    3

SQL> insert into mes_vols
  2 values (11,to_date('08/09/2004 20:30:00', 'DD/MM/YYYY HH24:MI:SS'),
  3         'Marquises', 1);
insert into mes_vols
*
ERREUR Ó la ligne 1 :
ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```



### 14.1.2 Supprimer une vue

La suppression d'une vue se fait en utilisant la commande `DROP` :

```
SQL> Drop view mes_vols;
Vue supprimée.
```

## 14.2 Les synonymes

Un synonyme est un nom logique donné à un objet existant, il peut être associé à un objet de schéma de type :

- ◆ Table
- ◆ Vue
- ◆ Sequence
- ◆ Cluster
- ◆ Procédure, fonction, package

Les tables sont visibles à condition de préfixer le nom des tables par le user de création des tables, par exemple la table `AVION` créée par le user `CHARLY` est accessible si on l'appelle par :  
`CHARLY.AVION`

Il est possible de créer des synonymes afin d'associer un nom à une table et de simplifier l'accès des tables aux utilisateurs.

Il s'agit de donner un autre nom à un objet afin de le référencer différemment.



Le paramètre `public` permet de rendre le synonyme accessible par tous.





Il est également possible d'associer des droits d'accès aux synonymes.

```
drop public synonym AVION ;
drop public synonym VOL ;

-- CREATION de SYNONYM pour base Tahiti
create public synonym AVION      for charly.AVION ;
create public synonym VOL        for charly.VOL ;
```

Pour pouvoir renommer un synonyme, il ne doit pas avoir été créé avec la clause `PUBLIC`. Si c'est le cas il faut le supprimer puis le re-crée.

```
RENAME [public] SYNONYM [schema.]anc_nom to [schema.]nouv_nom ;
```

Pour supprimer un synonyme on utilise la commande `DROP` :

```
DROP [public] SYNONYM (schema.)nom_synonym ;
```

### 14.3 Les séquences

Les séquences sont des objets permettant de gérer les accès concurrents sur une colonne de table et d'éviter les inter-blocages.

Par exemple le calcul automatique d'une clé primaire contenant un numéro séquentiel.

L'appel de la séquence lors de l'insertion des données permet de récupérer un numéro calculé par Oracle à chaque accès base. Ce numéro est utilisé comme identifiant et est unique.

Une seule séquence doit être créée pour chaque table de la base de données.

Il est possible d'associer un synonyme à la séquence avant de donner les droits d'utilisation de celle-ci aux « USERS ».



Une séquence concerne obligatoirement une colonne numérique.



### 14.3.1 Créer une séquence

La création d'une séquence se fait avec la commande `CREATE SEQUENCE` :

```
CREATE SEQUENCE NOM_SEQUENCE
    INCREMENT BY ENTIER
    START WITH ENTIER
    MAXVALUE ENTIER | NOMAXVALUE
    MINVALUE ENTIER | NOMINVALUE
    CYCLE | NOCYCLE
    CACHE ENTIER | NOCACHE
    ORDER | NOORDER
;
```

- `INCREMENT BY` : indique le pas d'incrément de la séquence
- `START WITH` : permet de spécifier la valeur de la première valeur de séquence à générer. Par défaut cette valeur correspond à `MINVALUE` pour une séquence ascendante et à `MAXVALUE` pour une séquence descendante.
- `MAXVALUE` : indique la valeur maximum de la séquence. Par défaut 10 puissance 27 pour l'ordre croissant et -1 pour l'ordre décroissant.
- `MINVALUE` : indique la valeur minimum de la séquence. Par défaut 1 (`NOMINVALUE`) pour l'ordre croissant et -10 puissance 26 pour l'ordre décroissant.
- `CYCLE` : permet de revenir à la valeur initiale en fin de limite. L'option `NOCYCLE` est prise par défaut.
- `CACHE` : spécifie au système d'allouer plusieurs séquences en même temps. La valeur spécifiée doit être inférieure au nombre de valeur du cycle. Oracle alloue par défaut 20 valeurs.
- `ORDER` : indique que les nombres doivent être générés dans l'ordre de la demande. `NOORDER` est l'option par défaut.

### 14.3.2 Utiliser une séquence

L'utilisation de `MA_SEQUENCE.NEXTVAL` permet de récupérer la valeur suivante attribuée par Oracle et de l'insérer dans la première colonne de la table client.

```
create sequence Ma_Sequence
    minvalue 100 ;

insert into client
    values ( Ma_Sequence.nextval, 'toto' ) ;
```



Pour rechercher la valeur courante il faut utiliser `CURRVAL` à la place de `NEXTVAL`.

```
select Ma_Sequence.currval  
from dual;
```

### 14.3.3 Modifier une séquence

La modification d'une séquence se fait en utilisant la commande `ALTER SEQUENCE`.

```
Alter sequence [schema.]sequence  
[increment by n]  
[start with n]  
[  
  {maxvalue n | nomaxvalue}  
  {minvalue n | nominvalue}  
  {cycle | nocycle}  
  {cache n | nocache}  
  {order | noorder} ] ;
```

- Les paramètres sont les mêmes que pour la création d'une séquence.
- `INCREMENT BY` : indique le pas d'incrément de la séquence
- `START WITH` : permet de spécifier la valeur de la première valeur de séquence à générer. Par défaut cette valeur correspond à `MINVALUE` pour une séquence ascendante et à `MAXVALUE` pour une séquence descendante.
- `MAXVALUE` : indique la valeur maximum de la séquence. Par défaut 10 puissance 27 pour l'ordre croissant et -1 pour l'ordre décroissant.
- `MINVALUE` : indique la valeur minimum de la séquence. Par défaut 1 (`NOMINVALUE`) pour l'ordre croissant et -10 puissance 26 pour l'ordre décroissant.
- `CYCLE` : permet de revenir à la valeur initiale en fin de limite. L'option `NOCYCLE` est prise par défaut.
- `CACHE` : spécifie au système d'allouer plusieurs séquences en même temps. La valeur spécifiée doit être inférieure au nombre de valeur du cycle. Oracle alloue par défaut 20 valeurs.
- `ORDER` : indique que les nombres doivent être générés dans l'ordre de la demande. `NOORDER` est l'option par défaut.

### 14.3.4 Supprimer une séquence

La suppression d'une séquence se fait en utilisant la commande `DROP SEQUENCE`.

```
Drop sequence [schema.]sequence ;
```



## 14.4 Procédures, Fonctions et Packages

Une procédure est une unité de traitement qui contient des commandes SQL relatives au langage de manipulation des données, des instructions PL/SQL, des variables, des constantes, et un gestionnaire d'erreurs.

Une fonction est une procédure qui retourne une valeur.

Un package est un agrégat de procédures et de fonctions.



Les fonctions, procédures et packages sont des programmes écrits en PL/SQL, langage de programmation dérivé de l'ADA interne au noyau Oracle. Ces programmes sont compilés et catalogués dans le dictionnaire de données.

Les packages, procédures, ou fonctions peuvent être appelés depuis toutes les applications qui possèdent une interface avec ORACLE (SQL\*PLUS, Pro\*C, SQL\*Forms, ou un outil client particulier comme NSDK par exemple).

Les procédures (fonctions) permettent de :

- ♦ Réduire le trafic sur le réseau (les procédures sont locales sur le serveur)
- ♦ Mettre en œuvre une architecture client/serveur de procédures et rendre indépendant le code client de celui des procédures (à l'API près)
- ♦ Masquer la complexité du code SQL (simple appel de procédure avec passage d'arguments)
- ♦ Mieux garantir l'intégrité des données (encapsulation des données par les procédures)
- ♦ Sécuriser l'accès aux données (accès à certaines tables seulement à travers les procédures)
- ♦ Optimiser le code (les procédures sont compilées avant l'exécution du programme et elles sont exécutées immédiatement si elles se trouvent dans la SGA (zone mémoire gérée par ORACLE). De plus une procédure peut être exécutée par plusieurs utilisateurs.

Les packages permettent de regrouper des procédures ou des fonctions (ou les deux). On évite ainsi d'avoir autant de sources que de procédures.

Le travail en équipes et l'architecture applicative peuvent donc plus facilement s'organiser du côté serveur, où les packages regrouperont des procédures choisies à un niveau fonctionnel. Les packages sont ensuite utilisés comme de simples bibliothèques par les programmes clients. Mais attention, il s'agit de bibliothèques distantes qui seront *processées* sur le serveur et non en locale (client/serveur de procédures).

Dans ce contexte, les équipes de développement doivent prendre garde à ne pas travailler chacune dans « leur coin ». Les développeurs ne doivent pas perdre de vue la logique globale de l'application et les scénarios d'activité des opérateurs de saisie.



## 14.5 Les Triggers

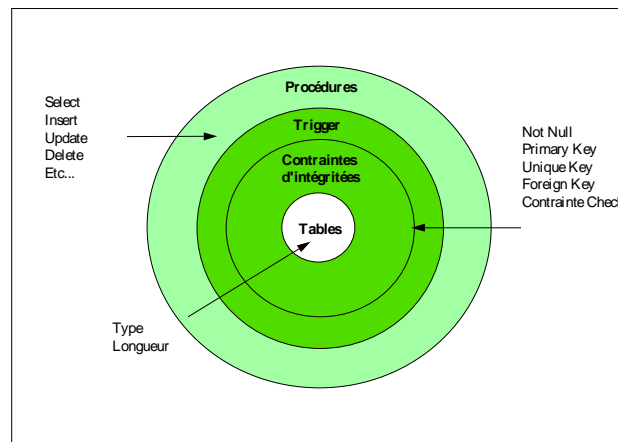
Un trigger permet de spécifier les réactions du système d'information lorsque l'on « touche » à ses données. Concrètement il s'agit de définir un traitement (un bloc PL/SQL) à réaliser lorsqu'un événement survient.

Les événements sont de six types (dont trois de base) et ils peuvent porter sur des tables ou des colonnes :

BEFORE	INSERT
AFTER	INSERT
BEFORE	UPDATE
AFTER	UPDATE
BEFORE	DELETE
AFTER	DELETE

Pour bien situer le rôle et l'intérêt des TRIGGERS, nous présentons ici une vue générale des contraintes sur le

Serveur :



*Vue générale des contraintes*

Les TRIGGERS permettent de :

renforcer la cohérence des données d'une façon transparente pour le développeur, mettre à jour automatiquement et d'une façon cohérente les tables (éventuellement en déclenchant d'autres TRIGGERS).



Rappelons que les contraintes d'intégrité sont garantes de la cohérence des données (pas de ligne de commande qui pointe sur une commande inexistante, pas de code postal avec une valeur supérieur à 10000, pas de client sans nom, etc. ...).

Les TRIGGERS et les contraintes d'intégrité ne sont pas de même nature même si les deux concepts sont liés à des déclenchements implicites.

Un trigger s'attache à définir un traitement sur un événement de base comme « Si INSERTION dans telle table alors faire TRAITEMENT ». L'intérêt du TRIGGER est double. Il s'agit d'une part de permettre l'encapsulation de l'ordre effectif (ici INSERTION) de mise à jour de la base, en vérifiant la cohérence de l'ordre. D'autre part, c'est la possibilité d'automatiser certains traitements de mise à jour en cascade.

Les traitements d'un TRIGGER (insert, update, delete) peuvent déclencher d'autres TRIGGERS ou solliciter les contraintes d'intégrité de la base qui sont les « derniers gardiens » de l'accès effectif aux données.

En version 9i il existe des TRIGGERS rattachés aux VUES ou des TRIGGERS sur événements systèmes.

En version 10g il est possible de gérer un commit ou un Rollback à l'intérieur du corps du trigger.

## 14.6 Les index

Les index permettent de retrouver rapidement les données d'une base. Ils peuvent être créés sur des tables ou des clusters. Ils interviennent donc directement dans les performances d'une base de données. Dans ORACLE, leur création est implicite sur les clés primaires (PRIMARY KEY) et unique (UNIQUE KEY).

L'utilisateur peut créer lui même ses propres index sur une ou plusieurs colonnes d'une table.

Un *index concaténé*, également appelé *index composé*, est créé sur plusieurs colonnes d'une table. Les colonnes ne doivent ni suivre forcément le même ordre que celui des colonnes de la table, ni être adjacents.

Un index comprend un maximum de 32 colonnes. Toutefois la taille totale des colonnes reste inférieure à un tiers de la taille du bloc de données.

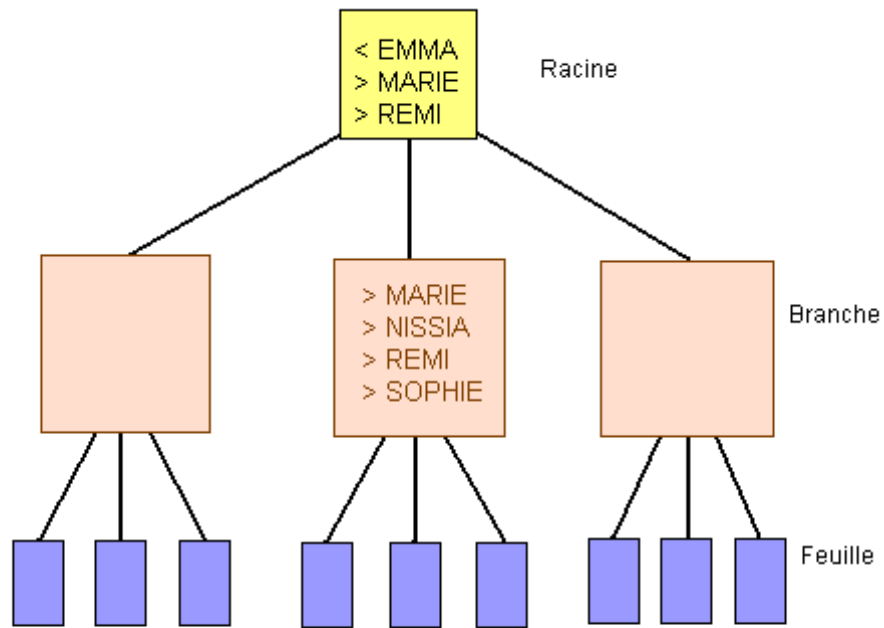


Un index unique garanti que deux lignes d'une table n'ont pas la même valeur dans la colonne qui le définit.  
Si trop d'index sont créés sur une table, ils pénalisent les performances.

Un index contient les valeurs des colonnes indexées et les adresses mémoires des lignes correspondants à ces colonnes.

Les index utilisent une structure analogue à celle des tables.

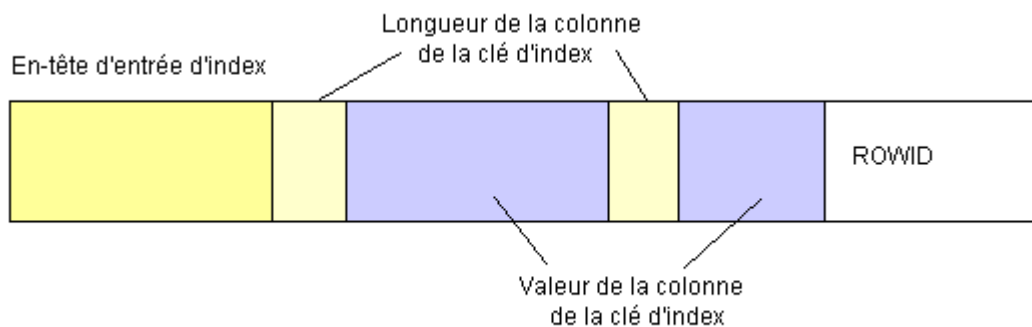




**Index organisés en B\*-Tree**

En termes algorithmiques, la recherche dans un B\*-Tree est semblable à celle réalisée dans un arbre binaire, à la différence qu'un arbre B\*-Tree peut contenir jusqu'à n nœuds enfant, alors qu'un nœud d'un arbre binaire ne peut en contenir que 2.

Oracle n'utilise pas d'index organisés en nœuds binaires mais plutôt une arborescence équilibrée.



**Format des feuilles d'index**

Les entrées d'index présentées ci-dessus ne concernent que les index globaux ou les tables non partitionnées.



Une entrée de feuille d'index se compose :

- ◆ Un entête d'entrée qui stocke le nombre de colonnes et les informations sur le verrouillage.
- ◆ Des éléments de contrôle pour stocker la longueur de la colonne d'index
- ◆ Les valeurs de la colonne d'index
- ◆ Le ROWID de la ligne qui contient les valeurs de la clé d'index

Les index réduisent les temps de réponse en lecture, mais pénalisent les performances en modifications (INSERT, UPDATE, DELETE).

Les index sont considérés comme des objets de la base. A ce titre, on peut les créer, les modifier, ou les supprimer. Comme pour une table, ou un utilisateur, la création d'un index nécessite des informations sur les caractéristiques de stockage (tablespace et paramètres).

La syntaxe de base de création d'un index est la suivante :

```
CREATE INDEX [unique] nom_index
ON          nom_table (liste des colonnes indexées)
TABLESPACE      nom_tablespace
```

```
-- =====
--      Index : A_POUR_PATRON_FK
-- =====
create index A_POUR_PATRON_FK on EMPLOYE (EMP_ID_EMP asc)
tablespace INDX
/
```



Si le nom du tablespace n'est pas présent c'est le tablespace par défaut de l'utilisateur qui sera utilisé.  
Si l'utilisateur n'a pas de tablespace de travail, c'est le tablespace systeme qui sera utilisé.  
Pour éviter cela, il ne faut pas donner de quota à l'utilisateur sur le tablespace système.





Règles d'utilisation d'un index B\*-Tree :

Indexer les colonnes fréquemment utilisées dans les clauses `WHERE`

S'assurer que les requêtes utilisant la clé d'index sont sélectives : moins de 5 à 15% des lignes de la table extraites (dépend de la répartition des données dans la table)

Privilégier les index concaténés

attention à l'ordre des colonnes

Ne pas hésiter à ajouter dans la clé d'index une colonne ramenée dans le `SELECT` plus d'accès à la table !

Indexer les clés étrangères

évite des problèmes de verrouillage sur la table enfant lors d'un `UPDATE` ou un `DELETE` sur la table père

Ne pas indexer les petites tables

Gérer les index uniques à l'aide des contraintes `PRIMARY KEY` ou `UNIQUE`

S'assurer que l'écriture des requêtes n'empêche pas l'index d'être utilisé

S'assurer que les index créés ne dégradent pas les performances des mises à jour

Les colonnes fréquemment utilisées dans les clauses `WHERE` peuvent l'être comme critère de sélection ou critère de jointure.

En général, une sélectivité inférieure à 5% est bonne et une sélectivité supérieure à 15% est mauvaise ; entre les deux, il faut tester ...

Pour la sélectivité, il faut que les valeurs de la colonne soient relativement uniques (beaucoup de valeurs distinctes) et que les conditions qui les utilisent soient elles-mêmes sélectives.

Parmi les colonnes candidates, il faut d'abord identifier les colonnes qui sont systématiquement présentes ensemble dans la clause `WHERE` : ce sont de bonnes candidates pour la création d'un index composé qui est généralement plus sélectif qu'un index simple.

L'ordre des colonnes est important dans un index composé : un index composé est utilisé si les colonnes de tête de la clé d'index sont présentes dans la condition (mais l'ordre des colonnes dans la condition n'a pas d'importance).

Indexer les petites tables ne sert à rien car, le nombre minimum de blocs à lire lors d'un accès par index est de 2 (1 bloc au minimum pour l'index et 1 bloc au minimum pour la table).

Grâce au paramètre `DB_FILE_MULTIBLOCK_READ_COUNT`, Oracle peut lire :

`DB_FILE_MULTIBLOCK_READ_COUNT` blocs en une entrée/sortie.

Donc, si la taille de la table est inférieure à `DB_FILE_MULTIBLOCK_READ_COUNT` blocs, un index est moins performant que le parcours complet.

Ainsi, en général, indexer des tables d'une vingtaine de blocs n'apporte rien.

Hormis les index uniques, il n'est jamais certain qu'un index soit réellement performant ; il faut donc tester !

Durant ces tests, il faut s'assurer que les index créés ne dégradent pas les performances des mises à jour.



### 14.6.1 Index et contraintes d'intégrité

Lorsqu'une contrainte de clé primaire ou de clé unique est créée ou activée, Oracle regarde s'il existe un index qui peut être utilisé pour vérifier la contrainte :

Index unique ou non unique, dont la clé est égale à, ou commence par, la clé de la contrainte

Si un tel index n'existe pas, Oracle crée un index unique pour vérifier la contrainte, dont la clé est égale à la clé de la contrainte.

L'option `USING INDEX` de la clause `CONSTRAINT` permet de spécifier les caractéristiques de stockage de cet index :

```
CONSTRAINT NOM_CONTRAENTE { PRIMARY KEY | UNIQUE }
(LISTE_COLONNES)
  [ USING INDEX
      [ TABLESPACE NOM_TABLESPACE ]
      [ PCTFREE VALEUR ]
      [ CLAUSE_STORAGE ] ]
```

Si la clause `USING INDEX` est spécifiée avec un tablespace différent du tablespace utilisé par l'index existant, Oracle retourne une erreur car il ne peut pas créer deux index sur la même clé.

Si deux index peuvent être utilisés, Oracle en choisit un.

```
-- =====
-- Table : EMPLOYE
-- =====
create table EMPLOYE
(
  ID_EMP      INTEGER          not null,
  NOM         VARCHAR2(30)     not null,
  SALAIRE     NUMBER(4)        not null,
  EMPLOI      VARCHAR2(20)     null   ,
  EMP_ID_EMP  INTEGER          null   ,
  constraint PK_EMPLOYE primary key (ID_EMP)
      using index
      tablespace INDX
)
tablespace DATA
/

-- =====
-- Index : CLES ETRANGERES
-- =====
alter table EMPLOYE
  add constraint FK_EMPLOYE_A_POUR_PA_EMPLOYE foreign key (EMP_ID_EMP)
      references EMPLOYE (ID_EMP)
      USING INDEX
```



```

TABLESPACE indx
PCTFREE 20
STORAGE ( INITIAL 2000K NEXT 400K MAXEXTENTS 64 PCTINCREASE 0 )
/

```

### 14.6.2 La clause USING INDEX

Elle permet de mentionner explicitement le nom d'un index existant à utiliser pour vérifier la contrainte :

```
USING INDEX NOM_INDEX
```

Elle permet d'inclure un ordre SQL CREATE INDEX pour créer explicitement l'index associé à la contrainte :

```
USING INDEX ( CREATION_INDEX )
```

- Ou création\_index est un ordre SQL CREATE [UNIQUE] INDEX classique.

```

SQL> alter table employe
2      add CONSTRAINT employe_nom_UNIQUE UNIQUE (nom)
3      USING INDEX
4      (
5      CREATE INDEX employe_nom ON employe(nom)
6      TABLESPACE indx
7      STORAGE (INITIAL 10M NEXT 10M PCTINCREASE 0)
8      )
9      ;

```

Table modifiée.

```

SQL> insert into employe values
2 (100, 'Martin',1000,'Prof',null);

```

1 ligne créée.

```

SQL> insert into employe values
2 (101, 'Martin',1000,'Prof',null);
insert into employe values
*
ERREUR Ó la ligne 1 :
ORA-00001: violation de contrainte unique (OPDEF.EMPLOYE_NOM_UNIQUE)

```

L'index mentionné ou créé peut être unique ou non unique.



### 14.6.3 *Suppression d'un index*

La suppression d'un index se fait en utilisant la commande.

```
DROP INDEX nom_index ;
```

Par défaut, avec Oracle9i, il n'y a pas de changement sur le sort de l'index associé à une contrainte lorsque cette dernière est supprimée :

Si l'index associé est unique, il est supprimé

Si l'index associé est non unique, il est conservé

Avec Oracle9i, il est possible d'indiquer explicitement si l'index associé à une contrainte supprimée doit être conservé ou supprimé.

```
ALTER TABLE DROP CONSTRAINT { NOM_CONTRAINTTE | PRIMARY KEY }  
KEEP INDEX | DROP INDEX  
;
```

A priori, conserver un index unique lors de la suppression d'une contrainte de clé primaire ou unique n'a pas de sens : l'unicité est toujours vérifiée ...



## 15 Complément sur les tables

Les tables sont des objets de stockage car elles utilisent de l'espace disque.

L'espace disque utilisé par les tables est appelé segment de table.

Ce sont des objets permanents.

### 15.1 Le Flash Back

La technologie de la mémoire d'Oracle 10g, offre la capacité d'interroger la version ancienne sur le schéma des objets, d'interroger les données historiques et d'améliorer les modifications d'analyses (d'effectuer des modifications d'analyse).

Toute transaction logique génère une nouvelle version de la base de données. Avec Oracle database 10g, vous pouvez naviguer à travers ces versions pour trouver une erreur et sa cause :

- ⑩ Flashback Query : interroge toutes les données historiques (dans le temps)
- ⑩ Flashback Version Query : voit toutes les versions des données modifiées et la transaction qui a effectué ce changement.
- ⑩ Flashback Transaction Query : voit tous les changements faits par une transaction.

La version 9i introduisait la notion de « *flashback query* » pour fournir un mécanisme simple pour réparer les erreurs humaines.

Oracle 10g étend la technologie *flashback* pour assurer vite et facilement une réparation à tous les niveaux :

- ◆ Flashback database ; vous laisse rapidement ramener votre base à un point dans le temps en réparant toutes les modifications apportées depuis cet instant.
- ◆ Flashback drop ; donne une solution pour restaurer accidentellement des tables
- ◆ Flashback table ; vous permet de retrouver rapidement une table et son contenu à un moment dans le passé.
- ◆ Flashback Query ; vous laisse voir les modifications apportée à une ou plusieurs données accompagnées de ses les méta-données.

Des évolutions importantes sur le comportement de la base de données apparaissent en version 10g. En effet avec l'extension des fonctionnalités du Flashback, Oracle conserve une version des tables supprimées dans la base de données :

- ⑩ la corbeille utilisée par le Flash Back Drop conserve une version de la table en la renommant.
- ⑩ Afin de supprimer définitivement une table, il convient de rajouter l'option `PURGE` lors du `DROP TABLE`.



En effet, vous pouvez visualiser tous les objets que vous avez supprimés à partir des vues :

USER\_RECYCLEBIN ou RECYCLEBIN.

DBA\_RECYCLEBIN contient tous les objets qui ont été perdus par tous les utilisateurs de la base de données et qui sont toujours dans la corbeille de recyclage.

La commande SQL\*Plus : SHOW RECYCLEBIN permet de visualiser les objets qui ne peuvent pas être supprimés.

```
SQL> connect charly/charly
Connecté.
SQL> show user
USER est "CHARLY"

-- avec option PURGE

SQL> select * from ma_table;

   COL_ID COL_NOM
-----
      1 clotilde ATTOUCHE
      1 melodie dupond

SQL> drop table ma_table purge;

Table supprimée.

SQL> SELECT original_name, object_name, type, ts_name
   2 FROM user_RECYCLEBIN;

aucune ligne sélectionnée

-- sans option PURGE

SQL> select * from ma_table;

   COL_ID COL_NOM
-----
      1 clotilde ATTOUCHE
      1 melodie dupond

SQL> drop table ma_table;
Table supprimée.

SQL> col original_name for A10
SQL> col type for A7
SQL> col ts_name for A7
SQL> SELECT original_name, object_name, type, ts_name
   2 FROM user_RECYCLEBIN;

ORIGINAL_N OBJECT_NAME                                TYPE      TS_NAME
-----
MA_TABLE   BIN$oUcVuIdeRXCa0dZtdQfaYg==$0 TABLE     USERS
```



La suppression d'une table se fait en utilisant la commande DROP.

```
SQL> drop table avion_2 ;
Table supprimée.

SQL> drop table vol;
drop table vol
      *
ERREUR Ó la ligne 1 :
ORA-02449: clés uniques/primaires de la table référencées par des clés
étrangères
```

Pour supprimer une table de façon définitive sans qu'elle aille dans la corbeille de recyclage il faut utiliser la commande :

```
DROP Table nom_table purge ;
```

### 15.1.1 Modifier une table par fusion : MERGE

L'ordre SQL MERGE permet de sélectionner des lignes dans une table en vue de les insérer ou de les modifier dans une autre table

Le tout en une seule opération

L'ordre SQL MERGE peut être utilisé en PL/SQL

```
MERGE INTO table_cible [alias] USING source [alias] ON (condition)
WHEN MATCHED THEN clause_update
WHEN NOT MATCHED THEN clause_insert ;

source table | vue | sous-requête

clause_update UPDATE SET colonne = expression | DEFAULT [... ]

clause_insert INSERT (colonne[,...]) VALUES (expression | DEFAULT [...])
```



- INTO table\_cible [alias] : spécifie la table cible des insertions ou mises à jour
  - table\_cible : nom de la table
  - alias : alias sur la table (optionnel)
- USING source [alias] : spécifie la source des données
  - source peut être une table, une vue ou une sous-requête
  - alias : alias de la source (optionnel)
- ON condition : définit la condition sur la table cible qui va déterminer la nature de l'opération effectuée sur chaque ligne de la table cible
  - Chaque ligne de la table cible telle que la condition est vraie est mise à jour avec les données correspondantes de la source
  - Si la condition n'est vérifiée pour aucune ligne de la table cible, Oracle insère une ligne dans la table cible avec les données correspondantes de la source
- WHEN MATCHED THEN clause\_update : spécifie l'ordre UPDATE qui est exécuté sur les lignes de la table cible lorsque la condition est vérifiée
  - UPDATE « normal » sans le nom de la table (déjà définie par la clause INTO de l'ordre MERGE)
- WHEN NOT MATCHED THEN clause\_insert : spécifie l'ordre INSERT qui est exécuté dans la table cible lorsque la condition n'est pas vérifiée
  - INSERT avec VALUES « normal » sans la clause INTO donnant le nom de la table (déjà définie par la clause INTO de l'ordre MERGE)

```
SQL> select v.id_avion, nom_avion, destination
2      from avion a, vol v
3      where v.id_avion = a.id_avion
4      and destination = 'Marquises'
5  ;

  ID_AVION NOM_AVION                DESTINATION
-----
          1 Caravelle                Marquises

SQL> select * from avion_2;

  ID_AVION NOM_AVION                DESTINATION
-----
          1 Caravelle                Marquises
          2                          Tahiti
          3                          Tahiti
          4                          Tahiti
          5                          Canaries
          6 Petit coucou

MERGE INTO avion_2 a                                -- cible
  USING (select v.id_avion, nom_avion, destination
        from avion a, vol v
        where v.id_avion = a.id_avion
        and destination = 'Marquises'
        ) d                                          -- source = requête => alias vm
  ON (a.id_avion = d.id_avion)                      -- en cas d'égalité
  WHEN matched then                                -- correspondance
```





```

        update set a.nom_avion = 'Essai Merge' -- mise à jour
    WHEN not matched then -- pas correspondance
        insert (a.nom_avion, a.destination) -- insérer
        values (d.nom_avion, d.destination)
;
1 ligne fusionnée.
SQL> select * from avion_2;

```

ID_AVION	NOM_AVION	DESTINATION
1	Essai Merge	Marquises
2		Tahiti
3		Tahiti
4		Tahiti
5		Canaries
6	Petit coucou	

Exemple sur l'égalité des identifiant d'avion.

```

SQL> select * from avion_2;

```

ID_AVION	NOM_AVION	DESTINATION
1	Essai Merge	Marquises
2		Tahiti
3		Tahiti
4		Tahiti
5		Canaries
6	Petit coucou	

```

SQL> select v.id_avion, nom_avion, destination
2     from avion a, vol v
3     where v.id_avion = a.id_avion
4     and destination = 'Marquises'
5 ;

```

ID_AVION	NOM_AVION	DESTINATION
1	Caravelle	Marquises

```

MERGE INTO avion_2 a -- cible
    USING (select v.id_avion, nom_avion, destination
        from avion a, vol v
        where v.id_avion = a.id_avion
        and destination = 'Marquises'
    ) d -- source = requête => alias vm
    ON (a.id_avion != d.id_avion) -- en cas d'égalité
    WHEN matched then -- correspondance
        update set a.nom_avion = 'Petit Coucou' -- mise à jour
    WHEN not matched then -- pas correspondance
        insert (a.nom_avion, a.destination) -- insérer
        values (d.nom_avion, d.destination)
5 lignes fusionnées.

```



Vérification :

```
SQL> select * from avion_2;

  ID_AVION NOM_AVION                                DESTINATION
-----
         1 Essai Merge                               Marquises
         2 Petit Coucou                              Tahiti
         3 Petit Coucou                              Tahiti
         4 Petit Coucou                              Tahiti
         5 Petit Coucou                              Canaries
         6 Petit Coucou

6 ligne(s) sélectionné(s).
```

En version 9i, la condition doit se faire sur l'identifiant sinon Oracle affiche une erreur :

```
ON (a.nom_avion != d.nom_avion)          -- en cas d'égalité
*
ERREUR Ó la ligne 7 :
ORA-00904: "A"."NOM_AVION" : identificateur non valide
```

### 15.1.2 Améliorations de la commande MERGE en version 10g

En version 10g, il y a deux nouveautés majeures pour la commande MERGE :

- ⑩ De nouvelles clauses et extensions pour l'utilisation standard de la commande MERGE, facilitant et accélérant son utilisation.
- ⑩ Une clause optionnelle DELETE pour la commande MERGE UPDATE.

#### Commande UPDATE et INSERT conditionnels

Vous pouvez ajouter une clause conditionnelle WHERE a une clause UPDATE ou INSERT d'une commande MERGE pour conditionner les opérations INSERT ou UPDATE.

```
-- Cet exemple montre l'utilisation d'une clause WHERE qui permet
-- aux paramètres UPDATE ou INSERT de pointer vers des produits 'non-obsolètes'.

MERGE
  Into product_change PC  -- destination table1
  USING products P        -- source/delta table
  ON (P.prod_id = PC.prod_id) -- join condition
  WHEN MATCHED THEN
    UPDATE
      SET PC.prod_new_price = P.prod_list_price
      WHERE P.prod_status <> 'obsolete'
  WHEN NOT MATCHED THEN
    INSERT (PC.prod_new_price)
    Values (P.prod_list_price)
```



```
| WHERE P.prod_status <> 'obsolete'
| ;
```

### Clause optionnelle DELETE

Vous pouvez utiliser la clause `DELETE` dans une commande `MERGE UPDATE` pour nettoyer les tables en les mettant à jour.

Seules les lignes affectées par la clause `DELETE` seront mises à jour par l'opération `MERGE` dans la table de destination.

La condition `WHERE` du `DELETE` évalue la valeur mise à jour, et non la valeur originale qui a été évalué par la condition `UPDATE SET`. Ainsi, si une ligne de la table de destination correspond à la condition du `DELETE` mais n'est pas incluse dans la jointure définie par la clause `ON`, elle n'est pas effacée.

```
| -- supprimer les lignes des produits dont le statut est devenu obsolète
| -- en effectuant l'UPDATE.
| -- elle supprime les produits obsolètes de la table de destination.
|
| MERGE
|   Into product_change PC -- destination table 1
|   USING products P      -- source/delta table
|   ON (P.prod_id = PC.prod_id) -- join condition
|   WHEN MATCHED THEN
|     UPDATE -- UPDATE IF JOIN
|       SET PC.prod_new_price = P.prod_list_price ,
|           PC.prod_new_status = P.prod_status
|     DELETE WHERE (PC.prod_new_status = 'obsolete') -- Purge
|   WHEN NOT MATCHED THEN -- INSERT IF NOT JOIN
|     INSERT (PC.prod_id, PC.prod_new_price, PC.prod_new_status)
|     Values (P.prod_id, P.prod_list_price, P.prod_status)
| ;
```

## 15.2 Créer une table à partir d'une table existante

La création d'une table peut se faire à partir d'une table existante en précisant la requête d'extraction des colonnes désirées.

```
| SQL> create table AVION_BIS
| 2 (id_avion, nom_avion)
| 3 as
| 4 select id_avion, nom_avion
| 5 from avion;
| Table créée.
|
| SQL> select * from avion_bis;
|
| ID_AVION NOM_AVION
| -----
```



```

1 Caravelle
2 Boeing
3 Planeur

```

### 15.3 Renommer une table

```

SQL> rename avion_bis to avion_2 ;
Table renommée.

SQL> select * from avion_bis;
select * from avion_bis
*
ERREUR Ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> select * from avion_2;

   ID_AVION NOM_AVION
-----
1 Caravelle
2 Boeing
3 Planeur

```

- La table AVION\_BIS est renommée en AVION\_2.

### 15.4 Les tables temporaires

Il est possible de créer des tables temporaires afin d'optimiser les temps d'exécution des requêtes, ou pour des raisons pratiques lors de traitements.

```

create global temporary table temp_chats
(
  nom    varchar2(15),
  owner  varchar2(10)
)
;

```

### 15.5 Les tables externes

Il s'agit de tables créées par un ordre SQL, `CREATE TABLE`, dont la définition est stockée dans la base (« méta-données »), mais dont les données sont stockées à l'extérieur de la base (dans des fichiers) et accessibles via un « driver » Oracle.

Le fonctionnement est complètement transparent du point de vue applicatif.

Par exemple elles permettent le stockage d'un fichier.



Seule la définition de la table est stockée dans la base, un peu comme une vue.

La définition d'une table externe comporte deux parties :

- ♦ Une partie qui décrit les colonnes (nom et type)
- ♦ Une partie qui décrit la correspondance entre les colonnes et les données externes

Les données externes peuvent avoir plus de champs, moins de champs, des types différents par rapport aux colonnes de la table.

Le driver est capable de les présenter telles qu'elles sont définies dans la table.

Oracle fournit un driver permettant d'accéder à des données stockées dans un fichier.



Il n'y a pas de clause de stockage, ni de clause « tablespace » pour ce type de tables.

Le driver Oracle Loader, utilise la technologie de SQL\*Loader.

Il permet d'accéder à des données stockées dans un fichier, avec une syntaxe proche de celle de SQL\*Loader pour spécifier la clause `ACCESS PARAMETERS`.

La vue `USER_EXTERNAL_TABLES` donne des informations spécifiques aux tables externes.

### Modification de tables externes

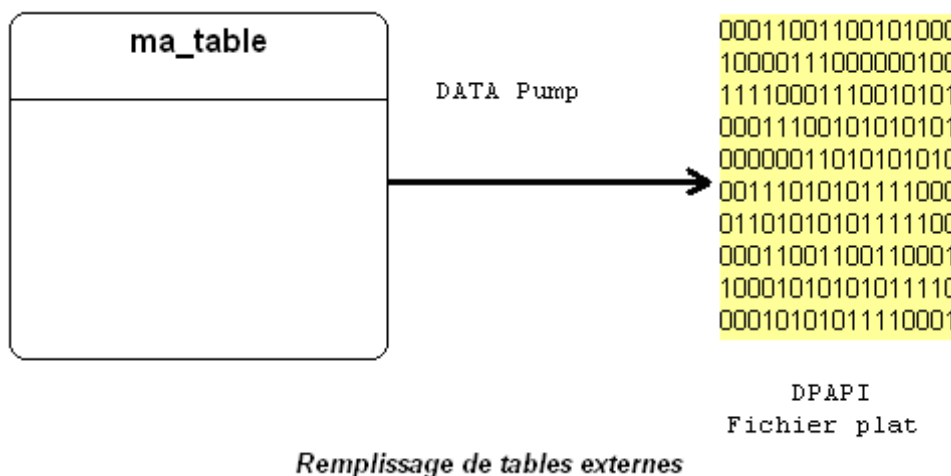
Dans les versions précédentes les tables externes étaient en lecture seule ; avec Oracle 10 g les tables externes peuvent être modifiées.

Il est possible d'utiliser la commande `CREATE TABLE AS SELECT` pour remplir une table externe composée de fichiers plats, avec le format propriétaire (`DIRECT PATH API`) indépendant du système d'exploitation ; même si les opérations `DML` ou de création d'index ne sont pas permises sur une table externe.

- ⑩ Le chargement de données fait référence à la lecture des données à partir d'une table externe qui sont ensuite chargées dans une table de la base de données.
- ⑩ Le déchargement des données fait référence à la lecture d'une table de la base et à l'insertion de données dans une table externe (ces opérations peuvent être utilisées avec des tables externes en utilisant le nouveau driver d'accès du *DATA Pump*).

L'avantage majeur de cette nouvelle fonctionnalité est de décharger des tables vers des fichiers plats et d'utiliser ces fichiers plats pour charger le système cible. De cette manière des volumes importants de données peuvent être transformés et chargés dans un fichier plat indépendant de la plate forme.





Quand les données sont extraites elles sont converties de la représentation interne d'Oracle vers une représentation externe native d'Oracle (DP&API). Ce processus est accompli via une commande `CREATE TABLE AS SELECT`. Dans ce cas la source correspond aux données extraites par la clause `SELECT` et la destination le driver d'accès Oracle `DATA Pump`.



L'opération de remplissage d'une table externe vers une table externe destinée à être utilisée avec le driver d'accès « *Oracle\_loader* » n'est pas possible.

Après la création et le remplissage d'une table externe via la commande `CREATE TABLE AS SELECT` aucune ligne ne peut plus être insérée, modifiée, ou supprimée.

Toute tentative de modification des données dans la table externe génère les erreurs `ORA-30657`.

Les fichiers de données créés pour les tables externes peuvent être déplacés pour être utilisés comme fichiers de données pour d'autres tables externes de la même base ou d'une base différente.

Ces fichiers ne peuvent être lus que via le driver d'accès Oracle `DATA pump`.

Lorsque des fichiers de données sont remplis via des tables externes différentes, ils peuvent être spécifiés dans la clause `LOCATION` d'une autre table externe. Ceci fournit une méthode d'agrégation de données à partir de sources multiples.

La seule restriction est que les meta-données pour toutes les tables externes doivent être exactement les mêmes.

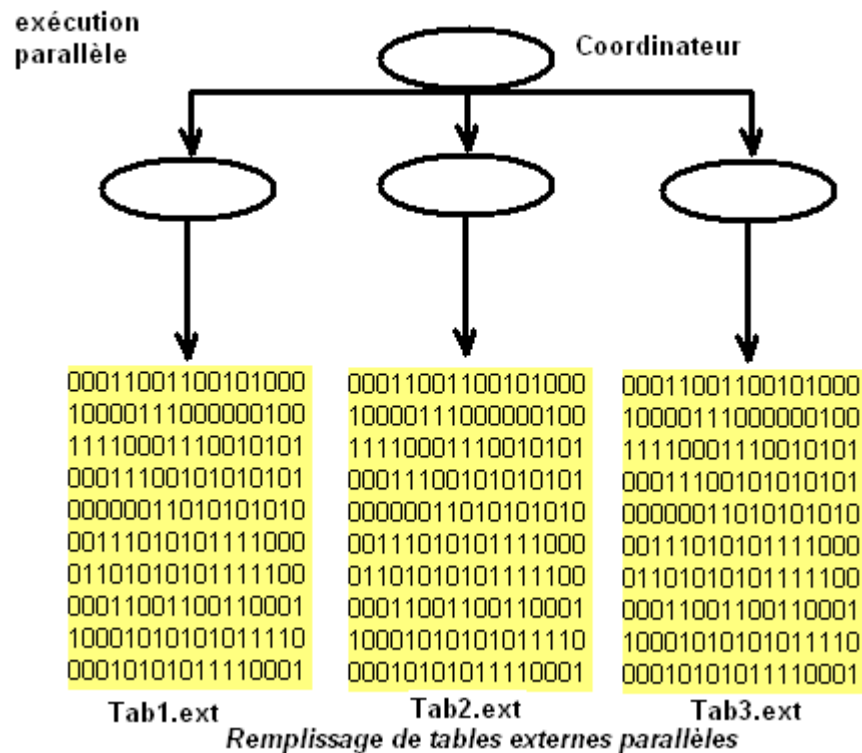
Le package `DBMS_META_DATA` peut être utilisé pour extraire les informations concernant les méta-données.

Ces opérations parallèles sont exécutées quand la table externe est déclarée comme `PARALLEL`.



Au contraire d'une requête parallèle sur une table externe le degré de parallélisme de remplissage parallèle est contraint par le nombre des fichiers concurrents qui peuvent être écrits par le driver d'accès.

Ainsi comme illustré dans le diagramme il n'y a jamais plus d'un seul serveur d'exécution parallèle qui écrit dans un fichier en simultanément.



Le nombre des fichiers dans la clause `LOCATION` doit correspondre aux degrés spécifiés de parallélisme car chaque processeur serveur d'I/O demande un fichier qui lui est propre. Tout fichier supplémentaire est ignoré.



S'il n'y a pas assez de fichiers pour le degré de parallélisme, le degré de parallélisme sera baissé pour correspondre au nombre de fichiers spécifiés dans la clause `LOCATION`.

```

CREATE TABLE emp_ext
  (first_name, last_name, department_name)
ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_DATAPUMP
  )
  
```



```

        DEFAULT DIRECTORY ext_dir
        LOCATION ('emp1.exp', 'emp2.exp', 'emp3.exp')
    )
PARALLEL
AS
SELECT e.first_name, e.last_name, d.departement_name
FROM employees e, departments d
WHERE e.departement_id = d.departement_id
      AND d.departement_name IN ('Marketing', 'Achats')
;

```

Cet exemple montre comment l'opération de remplissage de table externe peut aider à exporter d'une manière sélective des enregistrements qui résultent d'une jointure entre les tables `EMPLOYEES` et `DEPARTEMENTS`.

L'exemple illustre comment, tous les employés qui travaillent dans les départements « Marketing » et « Achats » peuvent être déchargés en parallèle.

On présume que le répertoire `EXT_DIR` a déjà été créé.

Dans la majorité des cas le driver d'accès « *Oracle\_datapump* » utilise les mêmes paramètres que le driver d'accès « *Oracle\_loader* ».

### Projection sur les tables externes

Dans le cas des fichiers de tables externes qui contiennent des lignes de données qui peuvent être rejetées à cause des formats de données la caractéristiques des données projetées permet d'obtenir des résultats cohérents indépendamment des colonnes au quelle fait référence la requête SQL qui accède à la table externe.

Avec les versions antérieures, seules les colonnes qui faisaient partie de la requête SQL étaient extraites par le driver d'accès.

Avec la version 10g vous pouvez spécifier la valeur `REFERENCED` pour l'attribut `PROJECT COLUMN` d'une table externe, afin de préciser au driver d'accès de traiter seulement les colonnes référencées. La valeur par défaut est `ALL`, qui précise au driver d'accès de traiter toujours toutes les colonnes d'une table externe.

Par exemple supposons que vous ayez défini une table externe constituée uniquement de champs numériques.

Suivant la valeur de l'attribut, `PROJECT COLUMN` vous pouvez obtenir des résultats différents pour les 2 commandes de l'exemple. En suivant le chemin gros tirets la requête de gauche retourne la valeur 2, alors que la requête de droite retourne la valeur 1, ceci a cause du fait que le driver d'accès a été paramétré pour extraire seulement 1 `ORDER_ID` dans le premier cas et seulement le `LINE_ID` dans le 2 eme cas.

Dans le 1ere cas les 2 lignes ont des `ORDER_ID` correctes. Dans le 2eme cas seulement la 1ere ligne a un `LIGNE_ID` correcte. Ainsi le 2eme `LIGNE_ID` est rejeté et la requête compte une ligne au lieu de 2.

En suivant le chemin (petits pointille) les 2 requêtes retournent la valeur 1. Ceci est du au fait que dans ce cas le driver d'accès a été paramétré pour extraire toutes les colonnes indépendamment de la sélection. Ainsi chaque colonne est convertie pour les 2 requêtes.





Oracle 10g permet de marquer une table externe comme `REFERENCED` ou `ALL` en fonction de vos besoins.

```
SELECT count(order_id)
FROM order_items_ext
;

SELECT count(line_id)
FROM order_items_ext
;

-- ORDER_ITEMS1.dat contient les valeurs :
2355,1,2289,46,200
2355,A,2264,50,100
```

La projection des colonnes `REFERENCED` est utile uniquement pour des raisons de performance car seules certaines colonnes seront interprétées (*parse*) et converties.

Si vous êtes sûr que les données sont correctes, alors l'option `REFERENCED` donnera de meilleurs résultats si un sous-ensemble des colonnes est sélectionné et fournira toujours le même résultat.

```
ALTER TABLE order_items_ext
PROJECT COLUMN {ALL | REFERENCED}
;
```

- `ALL` : valeur par défaut
- `REFERENCED` : plus performant et utilisé quand on est sûr que les données sont connues et valides

Il est possible de vérifier l'état de vos tables externes en regardant dans les nouvelles colonnes `PROPERTY` de la vue `DBA_EXTERNAL_TABLES` ; la valeur par défaut pour `PROPERTY` de la vue `DBA_EXTERNAL_TABLES` est « `project all column` ».



```
SELECT property  
FROM DBA_EXTERNAL_TABLES  
WHERE table_name = 'OWNER_ITEMS_EXT'  
;
```



## 16 Les vues Matérialisées

Vous pouvez utiliser les *vue matérialisées* pour fournir des copies locales de données distantes à vos utilisateurs ou pour stocker des données dupliquées dans la même base de données.

Une vue matérialisée se fonde sur une requête qui utilise un lien de base de données appelé `DATABASE LINK`, pour sélectionner des données dans une base distante.

Ces vues peuvent être implémentées en lecture (`READ-ONLY`) ou en écriture (`UPDATABLE`).

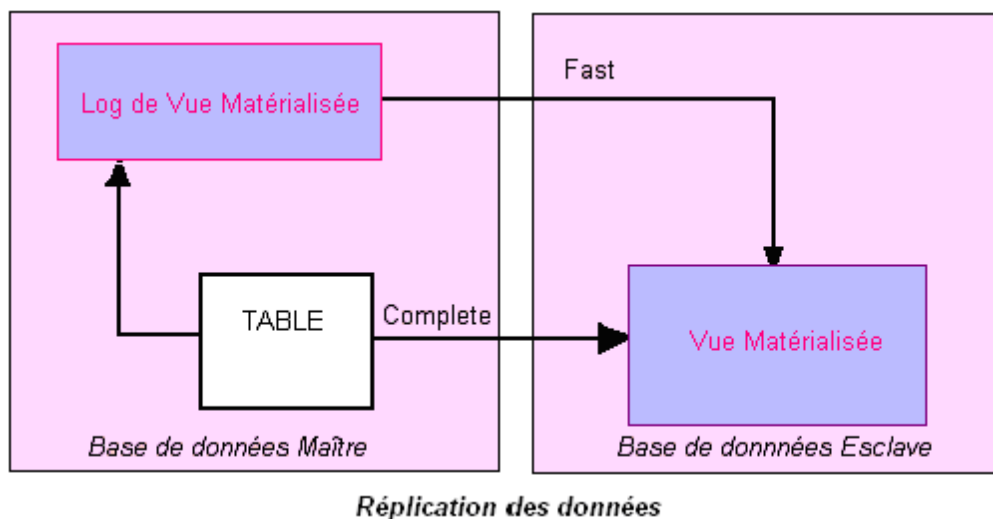
Il est également possible de les indexer.

Selon la complexité de la requête qui définit une vue matérialisée, il est possible d'utiliser un journal de vue matérialisée (*Materialized View Log*) afin d'optimiser les performances des opérations de réplication.

Les vues matérialisées peuvent être utilisées dans les *Datawarehouses* afin d'améliorer les performances en étant utilisées comme objet d'agrégat.

Ces vues sont alors utilisées par l'optimiseur Oracle (*CBO*) pour établir le plan d'exécution des requêtes.

Il est également possible de créer des vues matérialisées partitionnées et baser ces vues sur des tables partitionnées.



```

drop materialized view sn_vol;

create materialized view sn_vol
pctfree 0
tablespace DATA_VIEWS
storage (initial 20K next 20K pctincrease 0)
parallel
build immediate
refresh fast commit
enable query rewrite
  as select Id_vol, Nom_vol
     from vol v, avion a
     where v.id_avion = a.id_avion
     ;

```

Dans notre exemple la vue est rafraîchie chaque fois qu'une transaction est validée dans la table « maître » (COMMIT).

Cette vue est remplie lors de sa création (BUILD IMMEDIATE) et le chargement des données s'effectue en parallèle (PARALLEL).

La commande ALTER MATERIALIZED VUE permet de modifier les paramètres de stockage de la vue.

### Améliorations Oracle 10g

Dans les versions précédentes des bases Oracle, le rafraîchissement rapide des Vues Matérialisées (MV) avec jointures et agrégats (MAV) supportait les propriétés suivantes :

Auto-jointures,  
Vues (supposant qu'elles puissent être « aplaties »),  
tables distantes dans la clause FROM de la requête définissant la vue matérialisée.

Cependant ces caractéristiques n'étaient pas supportées dans les Vues Matérialisées (MV) utilisant des jointures (MJV).

La base Oracle 10g supporte maintenant le rafraîchissement rapide pour les MJV à condition que :

- ⑩ Si la MJV a des instances multiples de la table dans la clause FROM, des colonnes ROWID pour chaque instance doivent être présentes dans la liste SELECT de la MV, et le log MV doit contenir la colonne ROWID.
- ⑩ Si la MJV a des vues référencées dans la clause FROM, la base Oracle 10g doit être capable d'effectuer une fusion complète des vues (*Complete View Merging*). Après la fusion, la MV doit satisfaire toutes les conditions nécessaires pour le rafraîchissement rapide. En particulier, la liste SELECT de la MV doit contenir des colonnes ROWID pour toutes les tables présentes dans la clause FROM de la MV. Aussi, les logs MV sont requis pour toutes les tables de la base, et elles doivent contenir la colonne ROWID.
- ⑩ Si la MJV contient des tables distantes dans la clause FROM, toutes ces tables doivent être localisées sur le même site. Attention, la commande ON COMMIT n'est plus supportée avec les MV de tables distantes. Les logs MV doivent être présents sur le site distant pour chaque table de la MV, et les colonnes ROWID doivent être présentes dans la liste SELECT de la MV.





Si le paramètre d'initialisation `OPTIMIZER_FEATURES_ENABLE` est défini à 10.0.0 ou plus, alors le paramètre `ENABLE_QUERY_REWRITE` est défini à `TRUE` par défaut.

