COURS

SQL

SQL

SQL

1. INTRODUCTION

1.1. Définitions

Une **base de données** est un ensemble d'informations structurées.

Un **SGBDR** (Système de Gestion de Bases de Données Relationnel) est un logiciel qui permet de :

- stocker,
- consulter,
- modifier,
- supprimer

les données de la base de données.

Un **SGBDR** stocke les informations dans des tables.

1.1. Définitions (suite ...)

SQL (Strutured Query Language) :

- est le langage utilisé pour accéder aux données d'une base de données.

- est normalisé. C'est un standard adopté par l'ANSI (American National Standards Institute).
ANSI SQL89

- est un langage ensembliste (non procédural)

- est un langage « universel » utilisé par :

    * les administrateurs

    * les développeurs

    * les utilisateurs

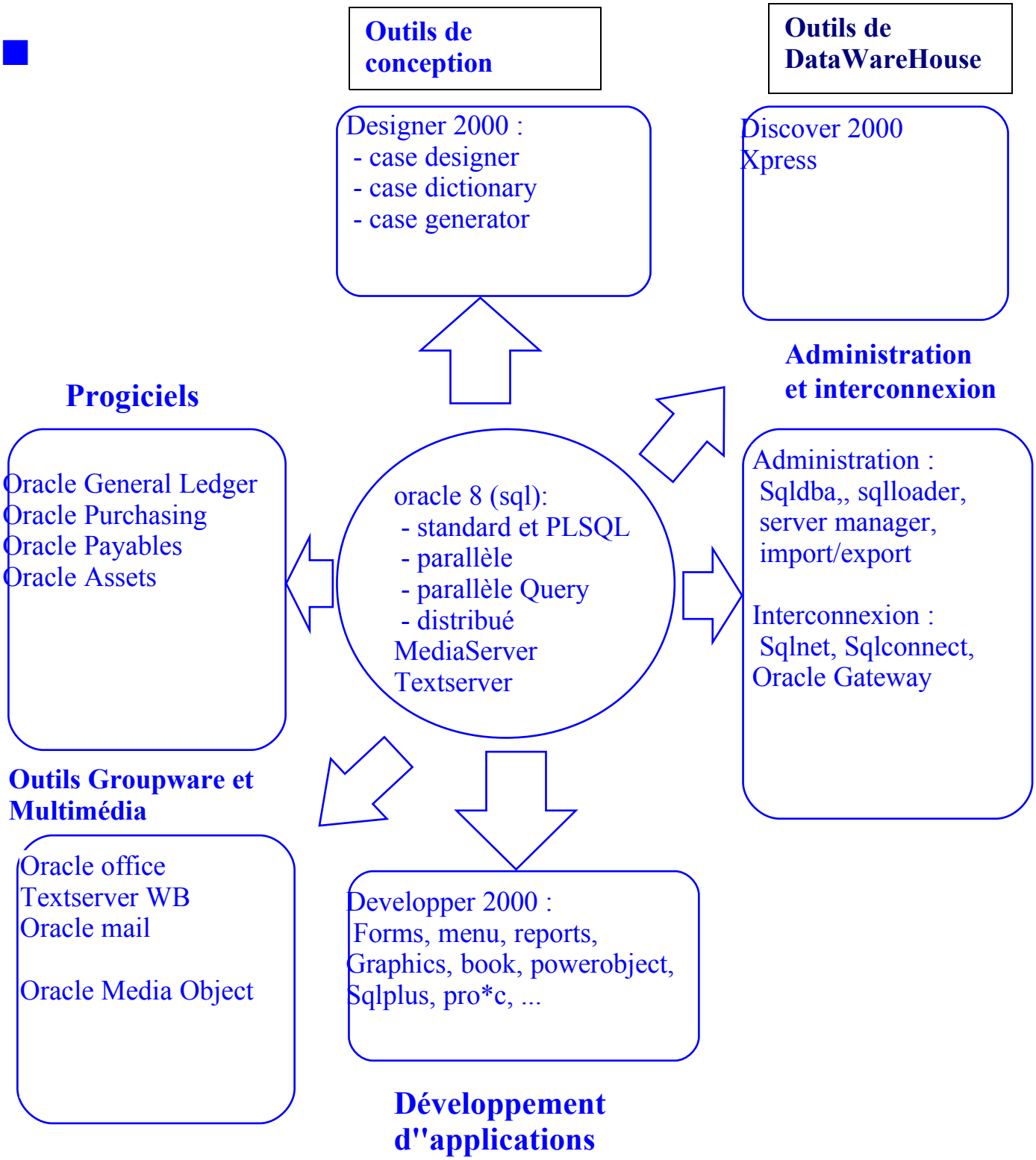
pour :

    * administrer et contrôler

    * définir et développer

    * manipuler

1.2. L'offre ORACLE



1.3. Les commandes

Commandes de manipulation des données :

- SELECT : interrogation
- INSERT : insertion
- UPDATE : mise à jour
- DELETE : suppression

Les **commandes de définition de données** :

- CREATE : création d'un objet
- ALTER : modification d'un objet
- TRUNCATE : supprimer les lignes d'une table
- DROP : supprimer un objet
- RENAME : renommer un objet

Remarque : les commandes GRANT et REVOKE seront vues dans le cours d'administration.

1.4. Les objets

Les objets du SGBD Relationnel ORACLE sont les suivants :

- les Tables,
- les Vues,
- les Index,
- les Séquences,
- les Synonymes,
- les Clusters.

Seuls les objets TABLES, VUES, INDEX et SYNONYMES seront vus dans ce cours.

2. Interrogation des données

2.1. Syntaxe du verbe SELECT

```
SELECT     [ALL | DISTINCT]  {[schéma.table].*
                | expr [c_alias], ...}

FROM   [schéma].obj [t_alias], [schéma].obj [t_alias], ...

[WHERE <condition>]

[CONNECT BY <condition>

      [START WITH <condition>]]

[GROUP BY expr, expr, ...

      [HAVING <condition>]]

[ORDER BY {expr|pos} [ASC|DESC],
                [{expr|pos} [ASC|DESC], ...]
```

2.1. Syntaxe du verbe SELECT (suite ...)

La clause : SELECT ...
FROM ...
WHERE ...

est une traduction simple du langage naturel. Elle permet de rechercher les données dans la base dans une ou plusieurs tables, dans une ou plusieurs vues.

Notes :

| : choix entre différentes options
{} : choix obligatoire
[] : facultatif

a) *obj* : peut être une TABLE, une VUE ou un SNAPSHOT

b) *expr* est une expression basée sur les valeurs d'une colonne

c) *c_alias* est le renommage de l'expression

d) *t_alias* est le renommage d'une table, vue ou shnapshot

2.2. Indépendance logique externe

L'objectif ici est de mettre en évidence l'indépendance logique externe.

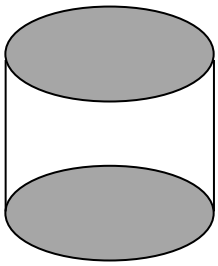


Table EMP (EMP#,NOM,ADR,SAL)

Table PROD(Prod#, PRIX)

...

?

PRIX	PROD#
...	...

N°	COUT	Prix Produit
...	coûte	...

EMP#	AUGM.
...	....

- Redisposition horizontaux des colonnes
- Renommage
- Constantes
- Calculs

- Calculs verticaux

TOTAL SAL
...

2.2. Indépendance logique externe (suite ...)

LE RENOMMAGE :

- alias d'attributs et

- alias des tables

Exemple :

```
SQL> SELECT p.pl#  num_pilote
     FROM pilote p;

NUM_PILOTE
--------------------
          1
          2
          3
          4
          5
          6
          8
          9
         10
         11
         12
         13
         14
         15
         16
         17

16 ligne(s) sélectionnée(s).
```

2.2. Indépendance logique externe (suite ...)

Exemple :

Ecrire de 3 manières différentes une projection sur toutes les colonnes de la table PILOTE.

SQL> SELECT * FROM pilote;

SQL> SELECT a.* FROM pilote a;

SQL > SELECT pilote.* from pilote;

Même résultat dans tous les cas :

```
PL# PLNOM        DNAISS   ADR                 TEL                SAL
--- ------------ -------- ------------------- ------------ ---------
  1 Miranda      16/08/52 Sophia Antipolis    93548254         18009
  2 St-exupéry   16/10/32 Lyon                91548254         12300
  3 Armstrong    11/03/30 Wapakoneta          96548254         24500
  4 Tintin       01/08/29 Bruxelles           93548254         21100
  5 Gagarine     12/08/34 Klouchino           93548454         22100
  6 Baudry       31/08/59 Toulouse            93548444         21000
  8 Bush         28/02/24 Milton              44556254         22000
  9 Ruskoi       16/08/30 Moscou              73548254         22000
 10 Mathé        12/08/38 Paris               23548254         15000
 11 Yen          19/09/42 Munich              13548254         29000
 12 Icare        17/12/62 Ithaques            73548211       17000,6
 13 Mopolo       04/11/55 Nice                93958211       17000,6
 14 Chretien     04/11/45                     73223322       15000,6
 15 Vernes       04/11/35 Paris                              17000,6
 16 Tournesol    04/11/29 Bruxelles                          15000,6
 17 Concorde     04/08/66 Paris                              21000,6

16 ligne(s) sélectionnée(s).
```

2.2. Indépendance logique externe (suite ...)

LA REDISPOSITION DES COLONNES (des attributs)

Exemple :

SQL> desc pilote;

```
 Nom                 Non renseigné  NULL?    Type
 ------------------------------- -------- ----
 PL#                                NOT NULL NUMBER(4)
 PLNOM                              NOT NULL CHAR(12)
 DNAISS                             NOT NULL DATE
 ADR                                         CHAR(20)
 TEL                                         CHAR(12)
 SAL                                NOT NULL NUMBER(7,2)
```

SQL> SELECT pl#, sal, tel,plnom
FROM pilote;

```
PL#        SAL TEL            PLNOM
--- --------- ------------ ------------
  1     18009 93548254     Miranda
  2     12300 91548254     St-exupéry
  3     24500 96548254     Armstrong
  4     21100 93548254     Tintin
  5     22100 93548454     Gagarine
  6     21000 93548444     Baudry
  8     22000 44556254     Bush
  9     22000 73548254     Ruskoi
 10     15000 23548254     Mathé
 11     29000 13548254     Yen
 12   17000,6 73548211     Icare
 13   17000,6 93958211     Mopolo
 14   15000,6 73223322     Chretien
 15   17000,6              Vernes
 16   15000,6              Tournesol
 17   21000,6              Concorde
```

2.2. Indépendance logique externe (suite ...)

LES CONSTANTES

On peut répéter une constante pour chaque ligne ramenée.

Les constantes sont de type numérique ou alphanumérique (entre ' ').

Exemple :

```
SQL> SELECT plnom NOM , 'gagne' GAIN ,
                    sal SALAIRE
        FROM pilote;

NOM          GAIN    SALAIRE
------------ ----- ---------
Miranda      gagne     18009
St-exupéry   gagne     12300
Armstrong    gagne     24500
Tintin       gagne     21100
Gagarine     gagne     22100
Baudry       gagne     21000
Bush         gagne     22000
Ruskoi       gagne     22000
Mathé        gagne     15000
Yen          gagne     29000
Icare        gagne   17000,6
Mopolo       gagne   17000,6
Chretien     gagne   15000,6
Vernes       gagne   17000,6
Tournesol    gagne   15000,6
Concorde     gagne   21000,6
```

LES CALCULS HORIZONTAUX

Le calcul horizontal fait intervenir une ou plusieurs colonnes d'une même table dans un tuple.

Exemple :

```
SQL> SELECT pl#, sal*12  "SALAIRE MENSUEL"
        FROM pilote;

PL# SALAIRE MENSUEL
--- ---------------
  1          216108
  2          147600
  3          294000
  4          253200
  5          265200
  6          252000
  8          264000
  9          264000
 10          180000
 11          348000
 12        204007,2
 13        204007,2
 14        180007,2
 15        204007,2
 16        180007,2
 17        252007,2

16 ligne(s) sélectionnée(s).
```

2.2. Indépendance logique externe (suite ...)

LES CALCULS VERTICAUX

Les calculs verticaux font intervenir les valeurs d'une colonne sur l'ensemble ou un sous-ensemble des tuples ramenés par une requête.

Remarque :
l'alias d'une colonne ou d'une expression sera de 30 caractères max. et sera entre "" si l'alias contient des séparateurs.

Exemple :

```
SQL> SELECT avtype TYPE,
                    SUM(cap) "CAPACITE TOTALE"
       FROM avion
       GROUP BY avtype;

TYPE       CAPACITE TOTALE
---------- ---------------
A300                  1300
A320                   320
B707                   400
B727                   250
Caravelle              300
Concorde               650

6 ligne(s) sélectionnée(s).
```

2.3. Elimination de doublons : DISTINCT

Le mot clé DISTINCT dans la clause SELECT :

- réalise un tri sur les colonnes et
- élimine les doublons.

Exemple :

SQL> SELECT DISTINCT avtype FROM avion;

```
AVTYPE
----------
A300
A320
B707
B727
Caravelle
Concorde
6 ligne(s) sélectionnée(s).
```

Il est possible de faire un DISTINCT de plusieurs colonnes.

Exemple :

SQL> SELECT DISTINCT avtype,cap FROM avion;

```
AVTYPE           CAP
---------- ---------
A300             300
A300             400
A320             320
B707             400
B727             250
Caravelle        300
Concorde         300
Concorde         350
```

EXERCICES Série 1

Alias des attributs

Ecrire la requête qui présente tous les pilotes de la compagnie avec le listing suivant:

Numéro Nom Adresse Salaire Mensuel

Redisposition des attributs

Ecrire la requête qui présente tous les pilotes de la compagnie avec le listing suivant

Nom Salaire Mensuel Numéro Adresse

Alias d'une table

Ecrire la requête qui renomme(alias) la relation PILOTE en P dans une requête.

Calculs horizontaux

Ecrire la requête qui calcule la durée d'un vol.

Ecrire une requête qui calcule le salaire annuel SAL_ANN, pour chaque pilote.

Calculs verticaux

Ecrire une requête qui calcule la somme des salaires des pilotes.

Distinct

Donner tous les types d'avions de la compagnie

2.4. Opération de sélection

SELECT ...

FROM ...

WHERE [NOT] prédicat1
[AND|OR]
[NOT] prédicat2 ...

La clause WHERE permet d'effectuer un filtrage de tuples. C'est à dire sélectionner un sous-ensemble de lignes dans les tables.

Seules les lignes vérifiant la clause WHERE seront retournées.

Prédicat :

nom de colonne constante expression	OPERATEUR	nom de colonne constante expression

- Les opérateurs logiques (AND, OR) peuvent être utilisés dans le cas de prédicats multiples.

- L'opérateur NOT inverse le sens du prédicat.

- Pas de limite dans le nombre de prédicats.

2.4. Opération de sélection (suite ...)

Exemples :

Lister tous les pilotes de la compagnie

```
SQL> SELECT *
       FROM pilote;
```

==> - pas de sélection
- tous les tuples de la relation PILOTE sont ramenés

Lister les pilotes qui vivent à Nice

```
SQL> SELECT *
       FROM PILOTE
       WHERE ADR='Nice';
```

==> - sélection : clause WHERE

- seuls les tuples de la relation PILOTE vérifant la clause WHERE sont ramenés

2.4.1. Opérateurs arithmétiques

Dans les critères de la clause WHERE, nous pouvons avoir les opérateurs de comparaison arithmétiques suivants :

= : égal,

!= : différent,

> : supérieur,

>= : supérieur ou égal,

< : inférieur,

<= : inférieur ou égal.

Exemple :

Liste des pilotes qui gagnent plus de 10000 et dont le numéro de tel est 93000000

```
SQL> SELECT plnom
        FROM pilote
        WHERE sal > 10000
           AND  tel='93000000';
```

2.4.2. Critères de comparaison : opérateurs sur les chaînes : LIKE et SOUNDEX

Opérateur LIKE

Caractères jokers de l'opérateur LIKE :

% : remplace 0 à n caractères

_ : remplace 1 et un seul caractère

Exemple 1 :

Sélectionnez les pilotes dont le nom commence par M.

```
SQL> SELECT *
     FROM pilote
     WHERE plnom LIKE 'M%';
```

Exemple 2 :

Sélectionnez les pilotes dont le nom contient un A en troisième position.

```
SQL> SELECT *  FROM pilote
     WHERE plnom LIKE '___A%';
```

2.4.2. Critères de comparaison : opérateurs sur les chaînes : LIKE et SOUNDEX (suite ...)

La clause ESCAPE permet de de-spécialiser les caractères jokers :

_

et

%.

Le caractère précisé derrière la clause ESCAPE permet la recherche des caractères _ et % dans une chaîne de caractères.

Exemple 3 :

Sélectionnez les pilotes dont le nom contient le caractère _.

```
SQL> SELECT *
     FROM pilote
     WHERE plnom LIKE '%*_%' ESCAPE '*';
```

2.4.2. Critères de comparaison : opérateurs sur les chaînes : LIKE et SOUNDEX (suite ...)

Opérateur SOUNDEX

SOUNDEX(chaîne) est une fonction qui permet une comparaison phonétique.

SOUNDEX(chaîne) génère une valeur numérique sur 4 octets (selon un algorithme précis).

Pour faire une comparaison phonétique entre 2 chaînes :

SOUNDEX(chaîne1) = SOUNDEX(chaîne2)

Exemple :

Sélectionnez les pilotes dont le nom ressemble à Tonton

```
SQL> SELECT plnom
        FROM pilote
     WHERE SOUNDEX(plnom) = SOUNDEX('Tonton');

PLNOM
------------
Tintin
```

2.4.3. Critères de comparaison avec l'opérateur IN

IN est l'opérateur qui permet de tester l'appartenance de la valeur d'une colonne à une liste.

Exemples :

Liste des vols dont la ville d'arrivée est Nice ou Paris.

```
SQL> SELECT vol#
        FROM vol
        WHERE va IN ('Nice ', 'Paris');
```

2.4.4. Critères de comparaison avec l'opérateur BETWEEN

BETWEEN est l'opérateur qui permet de tester si une valeur appartient à un intervalle.

Remarque : les bornes sont incluses.

Exemple :

Salaire et nom des pilotes gagnant entre 15000 et 18000

```
SQL> SELECT plnom, sal
        FROM pilote
        WHERE sal BETWEEN 15000 AND 18000;

PLNOM                SAL
------------ ----------
Mathé             15000
Icare           17000,6
Mopolo          17000,6
Chretien        15000,6
Vernes          17000,6
Tournesol       15000,6

6 ligne(s) sélectionnée(s).
```

2.4.5. Critères de comparaison avec une valeur nulle

IS NULL et IS NOT NULL sont les opérateurs qui permettent de tester si une valeur a été définie ou pas pour une colonne.
NULL : non défini.

SELECT ...
FROM table
WHERE coli **IS NULL**; coli non renseignée
ou SELECT ...
FROM table
WHERE coli **IS NOT NULL**; coli renseignée

Remarque : pour tester l'absence de valeur , ne pas utiliser = NULL ou != NULL.

Note : la syntaxe de comparaison est la suivante :
colonne IS NULL | IS NOT NULL

Exemple :
Nom des pilotes dont le numéro de tél. n'est pas renseigné

```
SQL> SELECT plnom
     FROM pilote
     WHERE tel IS NULL;
```

2.4.6. Les opérateurs ANY, SOME et ALL

Ils se combinent avec l'un des opérateurs arithmétiques :

{= | != | > | >= | < | <= } ANY : au moins 1 ...

SOME : au moins 1 ...

ALL : tout ...

Exemple 1 :
Sélectionnez les pilotes dont l'adresse est 'Nice' ou 'Paris'

```
SQL> SELECT plnom
        FROM pilote
        WHERE adr = ANY ('Nice', 'Paris');
```

Remarques :

- l'opérateur **ANY** est équivalent à l'opérateur **SOME**.

- la condition **=ANY** est équivalent à l'opérateur **IN**.

2.4.6. Les opérateurs ANY, SOME et ALL (suite ...)

Exemple 2 :

Sélectionnez les pilotes dont le salaire n'est pas un nombre rond.

```
SQL> SELECT plnom
        FROM pilote
        WHERE sal != ALL (12000, 13000, 14000, 15000,
16000, 17000, 18000, 19000, 20000, 21000, 22000, 24000,
25000, 26000, 27000, 28000,29000);
```

Remarque :

La condition **!= ALL** est équivalente à la condition **NOT IN**.

EXERCICES Série 2

"Numéros et type d'avions de capacité supérieure à 300"

"Nom des pilotes habitants Nice ou Paris"

"Quels sont les noms de pilotes comportant un 't' en quatrième position ou dont le nom se prononce 'Bodri'.

"Quels sont les vols au départ de Nice, Paris ou Bordeaux ?"

"Quels sont les avions dont la capacité est comprise entre 250 et 310 ?"

"Quels sont les pilotes dont l'adresse ou le téléphone sont inconnus ?"

"Nom des pilotes ayant un 'a' et un 'e' dans leur nom"

"Nom des pilotes ayant 2 'o' dans leur nom "

"Nom des pilotes dont le numéro de téléphone est renseigné"

2.5. Expressions et fonctions

L'objectif est de faire des calculs sur des :

- constantes,
- variables

de type :

- numériques,
- caractères,
- dates.

2.5.1. Les expressions

Colonne Constante Fonction	Opérateur	Colonne Constante Fonction

- Opérateurs arithmétiques : + - * /

- Opérateur sur chaînes de caractères : ||

- Pas d'opérateurs spécifiques aux dates.

Exemple1 :

Simuler une augmentation de 10% des salaires des pilotes

```
SQL> SELECT sal * 1.10 AUGMENTATION
        FROM pilote;
```

Exemple 2 :

Pilotes dont le salaire est supérieur à 2 fois 10000

```
SQL> SELECT *
        FROM pilote
        WHERE sal > 10 000 * 2;
```

2.5.1. Les expressions (suite ...)

Exemple 3 :

ajouter 3 jours à une date

'08-DEC-90' + 3 = '11-DEC-90'

Exemple 4 :

enlever 3 jours à une date

'11-DEC-90' - 3 = '08-DEC-90'

Exemple 5 :

nombre de jours entre 2 dates

date1 - date 2 = nbjours

2.5.1. Les expressions (suite ...)

Exemple 6 :

Noms et adresses des pilotes

```
SQL> SELECT plnom || '---->' || adr
        FROM pilote;
```

```
PLNOM||'---->'||ADR
-----------------------------------
Miranda      ---->Sophia Antipolis
St-exupéry   ---->Lyon
Armstrong    ---->Wapakoneta
Tintin       ---->Bruxelles
Gagarine     ---->Klouchino
Baudry       ---->Toulouse
Bush         ---->Milton
Ruskoi       ---->Moscou
Mathé        ---->Paris
Yen          ---->Munich
Icare        ---->Ithaques
Mopolo       ---->Nice
Chretien     ---->
Vernes       ---->Paris
Tournesol    ---->Bruxelles
Concorde     ---->Paris
scott        ---->Nice
Conficius    ---->Pekin

18 ligne(s) sélectionnée(s).
```

2.5.2. Les fonctions

Fonctions numériques

- ABS(n) : valeur absolue de n

- SIGN(n) : signe de n (-1 ou 0 ou +1)

- CEIL(n) : plus petit entier >= n

- FLOOR(n) : plus grand entier <= n

- MOD(m,n) : reste de la division de m par n

- POWER(m,n) : m élevé à la puissance n

- SQRT(n) : racine carrée de n (message d'erreur si n < 0)

- ROUND(n,[m]) : arrondi de n à 10-m

Ex : ROUND(125.2) = 125
ROUND(1600,-3)= 2000
ROUND(1100,- 3) = 1000
ROUND(345.343,2) = 345.34
ROUND(345.347,2) = 345.35

- TRUNC(n,[m]) : n tronqué à 10-m

Ex : TRUNC(2500,-3) = 2000
TRUNC(2400,-3) = 2000
TRUNC(345.343,2) = 345.34
TRUNC(345.347,2) = 345.34

2.5.2. Les fonctions (suite ...)

Fonctions caractères

- LENGTH(chaîne) : longueur de la chaîne

- UPPER(chaîne) : toutes les lettres de la chaîne en majuscules

- LOWER(chaîne) : toutes les lettres de la chaîne en minuscules

- INITCAP(chaîne) : première lettre de chaque mot de la chaîne en majuscules, les autres en minuscules)

- LPAD(chaîne,lg,[chaîne]) : compléter à gauche par une chaîne de caractères sur une longueur donnée.
Exemple :
LPAD('DUPOND',10,'*# ') = '*#*#DUPOND'

- RPAD(chaîne,lg,[chaîne]) : compléter à droite par une chaîne de caractères sur une longueur donnée.
Exemple :
RPAD('DUPOND',10,'* ') = 'DUPOND****'

Remarque : LPAD et RPAD peuvent tronquer une chaîne si lg < longueur totale de la chaîne.

2.5.2. Les fonctions (suite ...)

- LTRIM(chaîne[,caractères]) : suppression à gauche de caractères dans la chaîne.

Exemple :

LTRIM('DUPOND','DU ') = 'POND'

- RTRIM(chaîne[,caractères]) : suppression à droite de caractères dans la chaîne.

Exemple :

RTRIM('DUPOND','UD ') = 'DUPON'

- SUBSTR(chaîne,position[,longueur]) : extraction d'une chaîne à partir d'une position donnée et sur une longueur donnée

Exemple :

SUBSTR('DUPOND',2,3) = 'UPO'

- INSTR(chaîne, sous_chaîne,[,position[,n]]) : recherche de la position de la n ième occurence d'une chaîne de caractères dans une autre chaîne de caractères à partir d'une position données.

Exemple :

INSTR('DUPOND','D',1,2) = 6

2.5.2. Les fonctions (suite ...)

- REPLACE(chaîne,car[,chaîne]) : remplace un ensemble de caractères

Exemples :

REPLACE('TUTU','U', 'OU') = 'TOUTOU'
REPLACE('TATA','T') = 'AA'

- TRANSLATE(chaîne,car_source,car_cible) : trancodage de certains caractères par d'autres caractères dans une chaîne de caractères.

Exemples :

TRANSLATE(plnom,'AM','12') :
- le A est remplacé par 1
- le M est remplacé par 2

dans les noms des pilotes

- SOUNDEX(chaîne) : (voir 2.7.2)

- ASCII(chaîne) : donne la correspondance ASCII du premier caractère de la chaîne.

Exemple : ASCII('ADFGRSE') = 65

- CHR(n) : caractère correspondant à la valeur de n en ASCII.

Exemple : CHR(65) = 'A'

2.5.2. Les fonctions (suite ...)

Fonctions date :

- LAST_DAY(date) : dernier jour du mois d'une date donnée

- NEXT_DAY(date, jour) : date du prochain jour à partir d'une date donnée.

- ADD_MONTHS(date,n) : ajoute n mois à une date donnée.
- MONTHS_BETWEEN(date1,date2) : nombre de mois entre 2 dates.
- ROUND(date[,'precision']) : arrondi d'une date en fonction de la précision

Exemples : SYSDATE = '12-JUL-96'

ROUND(sysdate,'MM') = '01-JUL-96'
ROUND(sysdate + 4 ,'MM') = '01-AUG-96'
ROUND(sysdate,'YY') = '01-JAN-97'

- TRUNC(date[,'precision']) : truncature d'une date en fonction de la précision

Exemples :

TRUNC(sysdate,'MM') = '01-JUL-96'
TRUNC(sysdate + 4 ,'MM') = '01-JUL-96'
TRUNC(sysdate,'YY') = '01-JAN-96'

2.5.2. Les fonctions (suite ...)

Fonctions de conversion de types :

- TO_NUMBER(chaîne) : conversion d'une chaîne de caractères en nombre
Exemple : TO_NUMBER('567') = 567

- TO_CHAR(chaîne[,'format']) : conversion d'une expression (date ou numérique) en chaîne de caractères selon un format de présentation.

- TO_DATE(chaîne[,'format']) : conversion d'une chaîne en date selon un format.

Quelques formats numériques :

9	Affichage de cette valeur si elle est différente de 0
0	Affichage de zéros à gauche pour une valeur à zéro
$	Affichage de la valeur préfixée par le signe '$ '
,	Affichage de ',' à l'endroit indiqué
.	Affichage du point décima à l'endroit indiqué

Exemple : TO_CHAR(1234,'0999999') = 0001234

2.5.2. Les fonctions (suite ...)

Quelques formats de conversion de date :

TO_CHAR(date[,'format'])

FORMAT étant la combinaison de codes suivants:

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

Exemple :

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

==> 01 07 1996 10 : 24

2.5.2. Les fonctions (suite ...)

Pour avoir les dates en lettres utiliser les formats suivants :

YEAR	année en toutes lettres
MONTH	mois en toutes lettres
MON	nom du mois sur 3 lettres
DAY	nom du jour
DY	nom du jour sur 3 lettres
SP	nombre en toutes lettres
...	

Exemple :

TO_CHAR(SYSDATE,' « LE » DD MONTH YYYY « A » HH24 : MI')

==> LE 26 SEPTEMBER 1996 A 16 : 30

2.5.2. Les fonctions (suite ...)

Fonctions diverses :

NVL(expr,valeur)
==> Si expr IS NULL
Alors valeur
Sinon expr
Finsi

Exemple :

```
SQL> SELECT   NVL(sal,0)
         FROM pilote;
```

DECODE(expression, valeur1, result1,
[, valeur2, result2]
...
[,defaut]
==> Si expression = valeur1
Alors result1
Sinon Si expression = valeur2
Alors result2
Sinon defaut
Finsi
Finsi
Remarque : result1, result2, ... defaut peuvent être de types différents.

EXERCICES Série 3

"Lister les pilotes avec leur salaire tronqués au millier"

*"Lister les pilotes avec leur salaire. Pour ceux gagnat 17000,6
remplacer le salaire par '****' "*

"Sélectionner les pilotes et leur téléphone. Pour ceux dont le télephone n'est pas renseigné, mettre ? "

2.6. Les fonctions de groupe / utilisation de fonctions aggrégatives

Les **fonctions de groupe** sont les suivantes :

- AVG(expr) moyenne
- COUNT(expr) nombre
- MAX(expr) valeur maximim
- MIN(expr) valeur minimum
- STDDEV(expr) écart-type
- SUM(expr) somme
- VARIANCE(expr) variance

Remarques :

- les valeurs NULL sont ignorées.
- COUNT(*) permet de compter les lignes d'une table.

Exemple :

```
SQL> SELECT adr, AVG(sal), COUNT(sal), MAX(sal),
                 MIN(sal), STDDEV(sal),SUM(sal),
                 VARIANCE(sal)
       FROM pilote GROUP BY adr ;
```

2.7. Présentation du résultat trié selon un ordre précis

Un résultat peut être trié grâce à la clause ORDER BY

- de façon ascendante ASC ou

- descendante DESC.

Remarques :

- par défaut en Oracle le tri est toujours ascendant.

- 16 critères de tri maximum.

- dans un tri par ordre croissant les valeurs NULL apparaissent toujours en dernier

Exemple :

```
SQL> SELECT plnom, adr
        FROM pilote
        ORDER BY plnom;
```

2.8. Utilisation des pseudo colonnes ROWID, USER et SYSDATE

ROWID, USER et SYSDATE représentent respectivement

- l'adresse d'un tuple, composée de trois champs :
  * numéro de bloc dans le fichier,
  * le numéro de tuple dans le bloc et
  * le numéro de fichier),

- l'utilisateur courant d'Oracle

- la date système.

Ce sont entre autres des colonnes implicites de toute table d'Oracle.

Note : la table DUAL appartient à SYS. Elle possède une seule colonne DUMMY et une seule ligne avec pour valeur X. Cette table sert à sélectionner des constantes, des pseudo colonnes ou des expressions en une seule ligne.

Exemple :
SQL> select SYSDATE, USER FROM SYS.DUAL;

```
SYSDATE  USER
-------- ------------------------------
09/04/96 SCOTT
```

EXERCICES Série 4

"Ecrire une requête qui donne le salaire du pilote qui gagne le plus :
<valeur à calculer> "Max salaire Pilote "

"Quels sont les noms, l'adresse et le salaire des pilotes de la compagnie, triés en ordre croissant sur l'adresse, et pour une même adresse en ordre décroissant sur le salaire ? "

"Ecrire une requête qui recherche si l'utilisateur courant d'Oracle est un pilote ?"

"Ecrire une requête qui rend ROWID, USER, SYSDATE, Numéros de vol de tous les vols effectués à la date d'aujourd'hui par le pilote Numéro 4 ?". L'heure de départ et d'arrivée doivent apparaître dans la liste des colonnes de projection.

2.9. Requêtes multi-relations sans sous-requêtes : la jointure ou produit cartésien

L'objectif de la **jointure** est de ramener sur une même ligne le résultat des informations venant de différentes tables.

Décomposition de la jointure :

1. Sélection
2. Projection des colonnes des différentes tables (colonnes du SELECT + colonnes de jointure)
3. Prédicat de jointure
4. Projection des colonnes du SELECT

Remarques :
- dans le prédicat de jointure comme dans le SELECT, préfixer les attributs si il y a ambiguité.
- dans le prédicat de jointure, les alias des tables peuvent être utilisés.

L'objectif de l'**auto-jointure** est de ramener sur la même ligne le résultat des informations provenant de 2 lignes de la même table.

2.10. Requêtes multi-relations avec les opérateurs ensemblistes

L'objectif est de manipuler les ensembles ramenés par plusieurs SELECT à l'aide des opérateurs ensemblistes.

Les opérateurs ensemblistes sont :

- l'union : UNION,

- l'intersection : INTERSECT et

- la différence : MINUS

Principe :

SELECT ... FROM ... WHERE ... ==> ensemble
opérateur ensembliste
SELECT ... FROM ... WHERE ... ==> ensemble
opérateur ensembliste
SELECT ... FROM ... WHERE ... ==> ensemble
...
SELECT ... FROM ... WHERE ... ==> ensemble

[ORDER BY]

2.10. Requêtes multi-relations avec les opérateurs ensemblistes (suite ...)

Règles :

- même nombre de variables en projection

- correspondance du type

- colonne de tri référencées par numéro d'ordre

Résultat :

- les titres des colonnes sont ceux du premier SELECT

- la largeur de la colonne est celle de la plus grande largeur parmi les SELECT

- opération distincte implicite (sauf UNION ALL)

2.11. Sous-interrogations non synchronisée

Principe :

lorsque dans un prédicat un des 2 arguments n'est pas connu, on utilise les sous-interrogations.

```
SELECT ...
FROM ...
WHERE variable Op  ?
```

Le **?** n'étant pas connu, il sera le résultat d'une sous-requête.

Règle d'exécution :

c'est la sous-requête de niveau le plus bas qui est évaluée en premier, puis la requête de niveau immédiatement supérieur, ...

CAS 1 : sous-interrogation ramenant une valeur
On utilise les opérateurs =, >, ...

CAS 2 : sous-interrogation ramenant plusieurs valeurs
On utilise les ALL, IN, ANY, SOME.

Remarque :

une sous-interrogation peut ramener plusieurs colonnes. (on teste l'égalité ou l'inégalité).

2.11. Sous-interrogations (suite ...)

L'opérateur EXISTS :

la sous-interrogation ramène VRAI s'il existe au moins une ligne en réponse à la sous-interrogation, FAUX sinon.

L'interrogation principale s'exécute si VRAI.

Syntaxe :

```
SELECT ...
FROM ...
WHERE [NOT] EXISTS (SELECT ...)
```

2.12. La jointure externe

La jointure externe ("outer join") permet de ramener sur la même ligne des informations venant de plsuieurs tables ainsi que les lignes d'une des tables n'ayant pas de correspondant.

Cette fonctionnalité est directement offerte par SQL d'Oracle, en faisant suivre, dans la condition de jointure, la colonne de la table dont on veut les lignes non sélectionnées, par le signe (+). Ce signe signifie qu'un tuple supplémentaire ne contenant que des blancs (une valeur NULL dans chaque colonne) a été ajouté à la table concernée lorsque la requête a été lancée. Ce tuple "NULL" est ensuite joint aux tuples de la seconde table, ce qui permet de visualiser aussi les tuples non sélectionnés.
Le (+) est à mettre du côté de la clé étrangère puisque c'est uniquement de ce que côté qu'il peut ne pas y avoir de correspondant pour des clés primaires.

Contraintes :
on ne peut effectuer une jointure externe entre plus de deux tables dans une même clause SELECT. Autrement, l'opérateur de jointure externe (+) doit apparaître au plus une fois dans un predicat.

2.13. Sous-interrogations synchronisées

Principe :

lorsque dans un prédicat un des 2 arguments n'est pas connu, on utilise les sous-interrogations

SELECT ...
FROM ...
WHERE variable Op **?**

(voir plus haut)

Mais lorsque la valeur **?** est susceptible de varier pour chaque ligne, on utilise les **sous-interrogations synchronisées.**

Règles :

- le prédicat de la sous-interrogation fait référence à une colonne de l'interrogation principale
- si une table est présente dans les 2 select, la renommer

Exécution :

L'exécution de la sous-interrogation se fait pour chaque ligne de l'interrogation principale.

EXERCICES Série 5

Requêtes avec alias obligatoires (auto-jointure) et préfixage d'attributs(naming)

"Donner toutes les paires de noms de pilotes distincts, habitant la même ville"

Requêtes effectuant une jointure syntaxique

"Donner tous les noms des pilotes qui ont des noms d'avions ?"

"Ecrire la requête qui donne les noms des pilotes qui conduisent un A300 ou B727 ?".

"Tester la requête suivante :

```
(SELECT PILOTE#, VD, VA
FROM vol)
        INTERSECT
(SELECT AVION#, VD, VA
FROM VOL
);
```

Quel est sa signification en langage naturel ?

Sous-requêtes connectées par les opérateurs ANY, ALL, EXISTS, IN.

"Quel est le nom des avions dont la capacité est supérieure à la capacité de chaque avion localisé à Nice ?"

"Quel est le nom des avions dont la capacité est au moins égale à celle d'un avion localisé à Nice ? "

"Quel est le nom des pilotes assurant un vol au départ de Nice ?"

"Quel est le nom des pilotes assurant au moins un vol ?"

"Quel est le nom des pilotes dont le salaire est supérieure au salaire maximum de tous les pilotes effectuant un vol au départ de Paris ?"

Requêtes multi-relations avec sous-requêtes indépendantes

"Quels sont les noms des pilotes qui gagnent plus que le pilote nr. 5?"

"Donner le nom des pilotes, et pour ceux qui sont en service, la liste des numéros de vols qu'ils assurent ?"

2.14. La recherche hiérarchique

SQL*Plus permet la représentation et la manipulation de données ayant une structure hiérarchique.

Les lignes résultats sont ordonnées selon le parcours de l'arbre.

Le niveau de la hiérarchie dans lequel se trouvent les données concernées par la recherche peut-être accessible par la pseudo-colonne (attribut implicite de la table), LEVEL, jusqu'à 256 niveaux.

SELECT <colonne [, colonne, ...] >
FROM
[WHERE <Nom_table>]
CONNECT BY PRIOR <colonne 1> = <colonne2>
[**AND** <condition>]
[**START WITH** <Condition>] ;

Notes :
- START WITH : ligne(s) de départ de construction de l'arborescence (racine de l'arbre)
- CONNECT BY fixe le parcours de la hiérarchie (lien père-fils)
- PRIOR : colonne de départ

2.14. La recherche hiérarchique (suite ...)

Limites :

- une requête hiérarchique ne doit pas contenir de jointure

- une requête hiérarchique ne peut être effectuée sur une vue de jointure

- la clause ORDER BY est prioritaire à la clause CONNECT BY

Remarque :
ORACLE détecte les boucles éventuelles dans le CONNECT BY. L'ordre est interrompu et un message est envoyé à l'utilisateur.

Remarque sur le prédicat de sélection :

- si il est placé à l'extérieur de la clause CONNECT BY, il élimine certaines valeurs uniquement (le parcours de l'arborescence n'est pas interrompu)

- si il est placé à l'intérieur de la clause CONNECT BY, il élimine certaines valeurs et leurs dépendances (le parcours de l'arborescence est interrompu).

EXERCICES Série 6

"Quels sont les vols en correspondance (direct ou indirecte) au départ de Paris ?"

Note : - NICE ne doit pas être une escale de départ.

2.15. Le partitionnement

(voir 2.9.)

Le partitionnement permet de regrouper les lignes résultat en fonction des différentes valeurs prises par une colonne spécifiée.

```
SELECT     ...
FROM       <Nom_table> , ...
GROUP BY<Colonne> [, <colonne>, ...]
[HAVING <condition>] ;
```

La spécification de la clause GROUP BY entraîne la création d'autant de sous-tables qu'il y a de valeurs différentes pour la colonne de partitionnement spécifiée.

De même que la clause WHERE joue le rôle de filtre pour la clause SELECT, la clause HAVING joue le rôle de filtre pour la clause GROUP BY. L'exécution de la clause HAVING sera effectuée juste après celle du GROUP BY, pour sélectionner les sous-tables qui satisfont la condition spécifiée.

Contraintes :
- la colonne de partitionnement doit figurer dans la clause SELECT.
- un seul GROUP BY est autorisé par requête.
-pas de GROUP BY dans une sous-requête.

EXERCICES Série 7

"Pour chaque ville de localisation d'avions de la compagnie (sauf "Paris") donner le nombre, les capacités minimales et maximales d'avions qui s'y trouvent ?"

"Quels sont les pilotes (avec leur nombre de vols) parmi les pilotes N° 1, 2, 3 , 4 et 13 qui assurent au moins 2 vols ?"

"Quelle est la capacité moyenne des avions par ville et par type ? "

3. Mise à jour des données

L'objectif de ce chapitre est de se familiariser avec les commandes de mise à jour des données d'une base.

Commandes :

- d'insertion (INSERT),

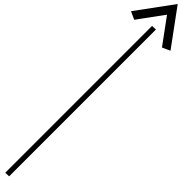
- de suppression (DELETE) et

- de mise à jour (UPDATE)

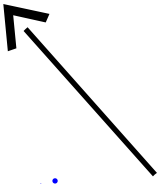
des données dans une base Oracle.

3.1. Insertion de lignes

INSERT INTO
<nom_user.nom_table | nom_user.nom_vue>
[(nom_colonnes[,nom_colonnes])]
VALUES (valeurs[,valeurs]) | sous_requête ;



Insertion par valeur



Insertion par requête

Remarque :
si toutes les valeurs des colonnes de la table sont insérées, il est inutile de préciser les colonnes. Si seules quelques valeurs sont insérées, préciser les colonnes.

Exemples :

```
SQL> insert into  pilote(pl#,plnom,dnaiss,sal)
          values(2, 'St-exupéry', '16/10/32',
12300.0);

SQL> insert into avion
          values(7, 'Mercure', 300, 'Paris', 'En
service');

SQL> insert into vol2
          select * from vol
         where vd='Paris';
```

3.2. Modification de lignes

UPDATE <[nom_user].nom_table | nom_vue>

SET nom_colonne1 = <expression1 | ordre_select>
[, nom_colonne2 = <expression | ordre_select> ...]

WHERE <critères_de_qualification>;

Exemple :

Augmenter les pilotes habitant Nice de 10%

```
SQL> UPDATE pilote
        SET sal = sal *1.10
        WHERE adr='Nice';
```

3.3. Suppression de lignes

3.3.1. Via la commande DELETE

DELETE FROM <nom_table | nom_vue>

[**WHERE** <critère_de_qualification>] ;

Remarque :
si pas de claure WHERE, la table entière est vidée.

Exemples :

Supprimer les pilotes habitant Nice

```
SQL > DELETE FROM pilote
        WHERE adr= 'Nice';
```

Supprimer tous les pilotes

```
SQL > DELETE FROM pilote;
```

3.3.2. Via la commande TRUNCATE

TRUNCATE TABLE nom_table

[DROP STORAGE | REUSE STORAGE]

Cette commande permet d'effectuer des suppressions rapides. C'est une commande du LDD d'Oracle et à ce titre équivaut à un commit.

Exemple :

SQL> TRUNCATE TABLE pilote;

Remarque :
Autre manière de supprimer les données d 'une table :
- la supprimer,
- la recréer

Exemple :
SQL> DROP TABLE pilote;
SQL> CREATE TABLE pilote(...);

3.3.2. Via la commande TRUNCATE (suite ...)

Avantages / Inconvénients des 3 solutions :

1ère option DELETE :

- la suppression avec DELETE consomme de *nombreuses ressources* : espace RedoLog, rollbck segment, ...

- pour chaque ligne supprimée, des *triggers* peuvent se déclencher

- la *place* prise par les lignes de la table n'est pas libérée. Elle reste associée à la table.

2ème option DROP :

- tous les index, contraintes d'intégrité et triggers associés à la table sont égelement supprimés

- tous les GRANT sur cette table sont supprimés

3.3.2. Via la commande TRUNCATE (suite ...)

3ème option TRUNCATE :

- truncate est plus rapide car cette commande ne génère pas d'informations (rollback) permettant de défaire cette suppression. L'ordre est validé (commit) de suite.

- truncate est irréversible pour la même raison.

- les contraintes, triggers et autorisations associés à la table ne sont pas impactés

- l'espace prise par la table et ses index peut être libéré (drop storage)

- les triggers ne sont pas déclenchés

EXERCICES Série 8

Effectuer des insertions respectivement dans pilote, avion et vol. Vérifier si les contraintes l'intégrités structurelles (entitité, domaine et de référence) sont prises en comptes. Vérifier aussi les valeurs nulles.

Note : insérer un pilote ayant votre nom de login oracle et 2 vols effectués par ce pilote.

Effectuer une insertion dans la table PILOTE2 via une sous-requête sur PILOTE.

Mettre à jour le salaire du pilote numéro 3 à 19000 F et Valider.

Supprimer le pilote numéro 11 et invalider.

Supprimer les lignes de la tables PILOTE2 via TRUNCATE. Tentez un ROLLBACK.

4. Le schéma de données

Les chapitres pécédents nous ont permis d'aborder l'aspect Manipulation de Données (*LMD*) du langage SQL.

Ce chapitre va nous permettre d'aborder l'aspect définition des données : le Langage de Définition de Données (*LDD*).

4.1. Définition du schéma : aspects statiques

4.1.1. Les types de données Oracle

CHAR(taille) : Chaîne - longueur fixe - de 1 à 255 octets

VARCHAR2(taille) : Chaîne de taille variable 1...2000 bytes

VARCHAR(taille) : Idem varchar2 (type réservé pour les versions futures d'Oracle : ne pas utiliser)

DATE : format par défaut JJ-MON-AA

LONG : type texte (taille jusqu'à 2Gbytes)

RAW(taille) : type binaire (taille de 1 à 255bytes)

LONG RAW : type binaire long (taille jusqu'à 2 Go)

NUMBER(n1[, n2]) :
n1 = nombre de digits du décimal (de 1 à 38)
n2 = nombre de digits après la virgule

ROWID : Chaîne hex. représentant l'adresse unique d'une ligne d'une table.

Remarque : une seule colonne de type LONG ou LONG RAW par table.

4.1.1. Les types de données Oracle (suite ...)

Comparaison varchar2 / char

* Comparaison 1:
Oracle ajoute des blancs au char le plus petit pour que les 2 chaînes aient la même longueur.
Oracle compare ensuite car. par car. Dès qu'il y a une différence il conclut lequel est plus grand.
==> utilisée si les 2 valeurs à comparer sont de type CHAR

* Comparaison 2 :
Oracle compare car. par car. Dès qu'il y a une différence il conclut lequel est plus grand. Si on a atteint la fin du 1er varchar sans rencontrer de différence avec le deuxième, c'est le plus long (le 2ème) qui est considéré comme le plus grand.
==> utilisée dès qu'une des 2 valeurs est de type varchar2 dans l'expression.

COMP1	COMP2
'ab' > 'aa'	'ab' > 'aa'
'ab' > 'a '	'ab' > 'a '
'ab' > 'a'	'ab' > 'a'
'ab' = 'ab'	'ab' = 'ab'
'a ' = 'a'	**'a ' > 'a'**

4.1.2. Création d'une table

CREATE TABLE <user>.<nom_table>
{(
<def_colonne1>,
<def_colonne2>,
...
<def_colonne2>

[contrainte_table]
)]
| as subquery};

Avec :

<def_colonne> :
nom_colonne type_colonne [DEFAULT expr]
[contrainte_col]

[contrainte_col] :
CONSTRAINT <nom_contrainte>

[NOT] NULL | UNIQUE |PRIMARY KEY |

REFERENCES [<user>].<table>(col...)
[ON DELETE CASCADE) |

CHECK (condition)

4.1.2. Création d'une table (suite ...)

[contrainte_table] :

CONSTRAINT <nom_contrainte>

UNIQUE (col...) | PRIMARY KEY (col...) |

FOREIGN KEY (col...) REFERENCES table (col...)
[ON DELETE CASCADE) |

CHECK (condition)

PRIMARY KEY vs UNIQUE

	P.K.	Unique
toutes les valeurs sont distinctes	oui	oui
la colonne est définie en NOT NULL	oui	pas oblig.
définit l'identifiant	oui	
précisé une seule fois par table	oui	
fait lien avec REFERENCES	oui	

4.1.2. Création d'une table (suite ...)

EXEMPLE :

Définition du schéma de la base de données aérienne.

```
create table pilote(
        pl#     number(4)    primary key,
        plnom   char(12)     not null unique,
        dnaiss  date         not null,
        adr     char(20)     default 'PARIS',
        tel     char(12),
        sal     number(7,4) not null
                             CHECK(sal < 70000.0)
    );

create table avion(
        av#     number(4)          primary key,
        avtype  char(10)
                CONSTRAINT chk_type
                CHECK(avtype IN ('A300', 'A310',
                    'A320', 'B707', 'Caravelle',
                    'B727', 'Concorde'),
        cap     number(4)          not null,
        loc     char(20)           not null,
        remarq  long
    );
```

4.1.2. Création d'une table (suite ...)

```
create table vol(
        vol#      number(4)    PRIMARY KEY,
        pilote#   number(4)
                  CONSTRAINT fk_pilote
                  REFERENCES PILOTE(PL#)
                       ON DELETE CASCADE,
        avion#    number(4)    NOT NULL,
        vd        char(20),
        va        char(20),
        hd        number(4)    NOT NULL,
        ha        number(4)
             CONSTRAINT ck_ha CHECK(HA > HD),
        dat       date,

        FOREIGN KEY (avion#)
        REFERENCES AVION(AV#)
    );
```

4.1.2. Création d'une table (suite ...)

Remarques :

- les mots clés PRIMARY KEY et REFERENCES permettent définir des contraintes d'intégrité d'entité et de référence. Sous Oracle 7 ils sont totalement supportés mais servent à la documentation sous Oracle 6 ;

- l'option DELETE CASCADE permet de propager les suppressions ;

- le mot clé FOREIGN KEY est identique à REFERENCES sauf qu'il s'applique au niveau table ;

- le mot clé CONSTRAINT permet de nommer explicitement les contraintes ;

- l'option CHECK permet de contrôler par exemple les contraintes d'intégrité de domaine ;

- la contrainte NOT NULL permet d'indiquer que la colonne doit toujours être renseignée ;

- la clause DEFAULT permet de fixer une valeur par défaut

- la clause UNIQUE permet d'éliminer les doublons (un index est implicitement créé idem pour primary key).

4.1.3. Création d'un index

Les index premettent d'accéder plus rapidement aux données.
Ils servent également à gérer l'unicité des clés primaires : un index UNIQUE est créé sur la ou les colonnes identifiant la clé primaire.

Les index sont stockés dans une structure externe à la table.
On peut créer plusieurs index sur une table.

Les index sont mis à jour par ORACLE lors des ordres INSERT, UPDATE, DELETE.

La création d'un index se fait grâce à la clause suivante :

CREATE [UNIQUE] INDEX nom_index

ON nom_table(colonne, colonne, ...) ;

Note : Dans Oracle V6, le seul moyen d'assurer l'intégrité d'entité est de créer un index unique sur la clé primaire. Les inconvénients sont : les risques de suppression et d'oubli de la création de l'index.

EXERCICES Série 9

"Créer une relation FORMATION, qui contiendra les renseignements suivants :
- le numéro de pilote ,
- le type de formation (ATT, VDN, PAI, ...)
- type d'appareil
- date de la formation "
Attention :
- un pilote à une date donnée participe à une formation
- un type d'appareil doit être : 'A300', 'A310', 'A320', 'B707' , 'Caravelle', 'B727' ou 'Concorde'
Créer la clé primaire (comme en V6 : sans utiliser la clause PRIMARY KEY) sur le numéro du pilote et la date de formation.

Créer un index unique sur le plnom de la table pilote que constatez vous

Créer également un index sur la colonne TYPEAPP de la table Formation

4.2. Définition du schéma : aspects dynamiques

4.2.1. Modification d'une table

ALTER TABLE [<nom_user>.] <Table> **...**

La clause **ALTER TABLE** permet :

* d'ajouter de nouvelles colonnes

ALTER TABLE [<nom_user>.] <Table>
ADD <def_col> ...

* d'ajouter de nouvelles contraintes d'intégrité

ALTER TABLE [<nom_user>.] <Table>
ADD <table_contrainte> ...

4.2.1. Modification d'une table (suite ...)

* de redéfinir une colonne
(type de données, taille, valeur par défaut)

ALTER TABLE [<nom_user>.] <Table>
MODIFY <def_col> ...

avec <def_col> :
(colonne[type_de_données] [DEFAULTexpr]
[col_contrainte]) ;

Note : NOT NULL est la seule contrainte pouvant être ajoutée par MODIFY.

* de modifier les paramètres de stockages (v. cours admin.)

* d'activer/désactiver/supprimer une contrainte d'intégrité
ALTER TABLE [<nom_user>.] <Table>
ENABLE <clause> | **DISABLE** <clause> |
DROP <clause> ...
avec <clause> :
UNIQUE (col1[,col2 ...]) [CASCADE] |
PRIMARY KEY [CASCADE] |
CONSTRAINT <nom_contrainte> [CASCADE]
* d'allouer explicitement des extensions de fichiers (v. adm)

4.2.1. Modification d'une table (suite ...)

Restrictions aux modifications des tables

AJOUT

- on peut ajouter une colonne de type NOT NULL uniquement si la table est vide

- on peut ajouter une contrainte uniquement au niveau table

MODIFICATION

- on peut retrécir une colonne uniquement si elle est vide

- on peut passer une colonne de NULL autorisé à NOT NULL uniquement si la colonne ne contient pas de valeur NULL

- on ne peut modifier une contrainte

SUPPRESSION

- on ne peut supprimer une colonne

- on peut supprimer une contrainte par son nom

4.2.1. Modification d'une table (suite ...)

Commentaires sur les tables ou les colonnes

Le commentaire sur une colonne se fait par la clause SQL suivante :

COMMENT ON
TABLE nom_table |
COLUMN table.colonne IS chaîne ;

Note : les commentaires sont insérés dans le Dictionnaire de Données.Leur consultation se fait entre autre à travers la vue USER_COL_COMMENTS.

Exemple :

```
SQL> COMMENT ON COLUMN pilote.pl#
          IS 'Numéro identifiant le pilote';
```

Pour supprimer un commentaire :

```
SQL> COMMENT ON COLUMN pilote.pl#
          IS '';
```

4.2.1. Modification d'une table (suite ...)

Consultation de la structure d'une table

Clause de listage des colonnes d'une table :

DESC[RIBE] [user.]nom_table ;

La clause DESCRIBE permet de lister les colonnes d'une table. L'utilisateur doit être propriétaire de la table ou en avoir reçu les droits.

Exemples :

SQL> DESC pilote;

SQL> DESCRIBE vol;

4.2.1. Modification d'une table (suite ...)

Synonyme d'une table

CREATE [PUBLIC] SYNONYM
[<user>.]<nom_synonyme>
FOR [<user>.]<nom_table> ;

Un synonyme est utilisé pour la sécurité et la facilité de manipulation.

ATTENTION : son utilisation abusive augmente le temps d'exécution des requêtes.

Notes :
- [Public] : le synonyme est accessible par tous les users.
- sert à référencer les objets sans indiquer leur propriétaire
- sert à référencer les objets sans indiquer leur base
- fournit un autre nom à un objet : alias
- un synonyme privé doit avoir un nom distinct dans le schéma d'un utilisateur
- un synonyme public peut avoir le nom de la table dans son schéma.

Remarque :
on peut également créer des synonymes pour des vues, séquences, procédures, ... et même synonymes.

EXERCICES Série 10

"Ajouter la colonne AGE à la table PILOTE. Un pilote doit avoir entre 25 et 60 ans.

"Ajouter une contrainte d'intégrité de référence au niveau table à la relation FORMATION (colonne PILOTE)"

"Modifier la colonne PL# de la table PILOTE en number(5).

Ajouter une valeur par défaut à la colonne VD dans VOL.

"Associer à l'attribut SALAIRE d'un pilote un commentaire puis s'assurer de son existence. Comment supprime - t-on un commentaire ?"

"Consulter la liste des colonnes de la table FORMATION"

"Attribuer un synonyme "Conducteurs" à la table PILOTE.

4.3. Le dictionnaire de données

Chaque base de données Oracle possède un dictionnaire de données :
il répertorie tous les objets de la base et leur définition.

Le dictionnaire de données est un ensemble de tables dans lesquelles sont stockées les informations sur les schémas des utilisateurs.

Le propriétaire des tables systèmes sous Oracle s'appelle SYS.

Le dictionnaire de données est mis à jour dynamiquement par ORACLE.

Un des avantages des bases de données relationnelles est que l'accès aux informations du dictionnaire se fait à travers la clause SELECT-FROM-WHERE.

Pour faciliter la consutation via SQL, il existe des vues et des synonymes systèmes

ATTENTION, l'utilisateur n'accèdera aux informations que sur ses objets ou ceux sur lesquels il à les GRANTS nécessaires.

4.3. Le dictionnaire de données (suite ...)

Les tables de bases

- seg$: segments définis dans la base de données

- obj$: objets définis sur la base de données

- tab$: tables définies dans la base de données y compris les clusters

- ind$: index définis dans la base de données

- col$: colonnes définies dans la base de données

- ts$: tablespaces définis dans la base de données

- ...

Notes :

- tables accessibles uniquement par l'utilisateur SYS.
- tables se terminant par un $
- il est interdit de mettre à jour directement ces tables.

4.3. Le dictionnaire de données (suite ...)

Les vues

- accessible_tables : contient les tables et vues accessibles par l'utilisateur

- all_catalog : tables, vues , synonym, ... accessibles par le user

- all_tab_columns : synonyme de la table *accessible_table*

- all_tab_grants : synonyme de la table table_privileges

- all_tables : description des tables accessibles par un user

- all_users : informations sur l'ensemble des users d'une base
- all_views : textes des vues accessibles par un utilisateur

- ...

- dba_catalog : toutes les tables, les vues, synonymes et séquences de la base

4.3. Le dictionnaire de données (suite ...)

- dba_data_files: informations sur les fichiers de la base de données

- dba_free_space : espace restant libre dans les fichiers de la base

- dba_users : informations sur l'ensemble des users de la base

- dba_tables : informations sur l'ensembles des tables de la base

- dba_views : texte de toutes les vues de la base

- ...

- user_catalog : tables, vues, ... dont l'utilisateur est propriétaire

- user_free_space : Extent libre dans un tablespace accessible par le user

- user_indexes : Descriptions des index de l'utilisateur

- user_tables : tables dont l'utilisateur est propriétaire

4.3. Le dictionnaire de données (suite ...)

- user_views : textes des vues de l'utilisateur

- user_users : info sur le user courant

- ...

Notes :
USER_* sont des vues donnant des informations sur les objets dont l'utilisateur est propriétaire

ALL_* sont des vues donnant des informations sur les objets auxquels l'utilisateur a accés

DBA_* sont des vues sur tous les objets de la base

- les vues commençant par dba_ sont accessibles par le DBA

- les vues commençant par all_ et user_ sont accessibles le DBA et l'utilisateur.

4.3. Le dictionnaire de données (suite ...)

Les synonymes

- cat : synonyme de la vue user_catalog
- clu : synonyme de la vue user_clusters
- cols : synonyme de la vue user_tab_columns
- dict : synonyme de la vue DICTIONARY
- ind : synonyme de la vue user_indexes
- seq : synonyme de la vue user_sequences
- syn : synonyme de la vue user_synonyms
- tab : synonyme de la vue user_tables

...

Les tables dynamiques

- v$process : informations sur les processus en cours
- v$bgprocess : descriptions des processus d'arrière plan
- v$licence : informations sur la validité des licenses
- v$lock : informations sur les verrous et les ressources
- v$parameter : informations sur les valeurs actuelles des paramètres
- v$session : informations sur les sessions courantes
- v$transaction : informations sur les transactions en cours
- ...

Note :

- ce sont les tables dynamiques de gestion des performances
- ces tables commences par un v$.

EXERCICES Série 11

"Quels sont les noms des colonnes de la table VOL ?"

"Quels sont les tables et les vues de votre schéma ?"

Notes : -col ou cols est un synonyme de user_tab_columns
-cat est un synonyme de user_catalog
-Tabletyp est le type de la colonne (une table, une vue...)

"Quelles sont les tables qui contiennent une colonne PLNUM ?"

"Quelles sont les vues du dictionnaire d'Oracle (voir DICT ou DICTIONARY) ? "

"Quels sont les tables appartenant à l'utilisateur SCOTT ?"

"Quels sont les contraintes existant dans votre schéma pour la table PILOTE ?"

4.4. Autres objets

Signalons l'existence d'autres objets pouvant appartenir au schéma d'un utilisateur :

- les séquences : servent à générer automatiquement les clés

- le database link : lien ers le schéma d'un utilisateur distant

- le shnapshot : copy asynchrone d'une table distante

- le trigger : alerte

- procédures et packages : procédures stockées

- le cluster : jointure physique entre deux ou plusieurs tables.

Ces objets seront traités dans le cours administration.

5. Concurrence d'accès

5.1. Transaction

Définition

Une transaction (unité logique de traitement) est une séquence d'instructions SQL qui doivent s'exécuter comme un tout.

5.1. Transaction (suite ...)

Début et fin d'une transaction Oracle

Une transaction ***débute*** :

- à la connexion à un outil

- à la fin de la transaction précédente.

Une transaction SQL ***se termine*** :

- par un ordre COMMIT ou ROLLBACK

- par un ordre du LDD valide :
CREATE, DROP, RENAME, ALTER, ...
==> La transaction est validée : COMMIT ;

- à la déconnexion d'un outil : DISCONNECT, EXEC SQL, RELEASE).
==> La transaction est validée : COMMIT ;

- lors d'une fin anormale du processus utilisateur.
==> La transaction est invalidée : ROLLBACK.

5.1. Transaction (suite ...)

Contrôle du déroulement d'une transaction

Les clauses de contrôle du déroulement des transactions sont :

```
COMMIT [WORK]
SAVEPOINT savepoint_id
ROLLBACK [WORK] [TO savepoint_id]
```

COMMIT :
- valide l'ensemble des modifications depuis le début de la transaction
- libère les verrous

ROLLBACK :
- restitue les données à leur valeur de début de transaction
- libère les verrous

ROLLBACK TO SAVEPOINT :
1. Pose d'étiquette : SAVEPOINT nom
2 . Annulation partielle :

```
ROLLBACK TO [SAVEPOINT] nom
```

Note : - l'utilisation de WORK est facultative
- le paramètre SAVEPOINT dans *init.ora* fixe le nombre de points de sauvegardes :
"savepoints" (par défaut 5)

EXERCICES Série 12

T1 : INSERT INTO pilote
values(18, 'Conficias', '19-SEP-42', 'Pekin', '13548254', 39000.0,null);
COMMIT ;

T2 : UPDATE pilote SET plnom='Conficios' WHERE plnom='Conficias';
ROLLBACK ;

T3 : UPDATE pilote SET plnom='Conficies' WHERE plnom='Conficias';
SAVEPOINT updt_conf1;

UPDATE pilote SET plnom='Conficius' WHERE plnom='Conficies';
SAVEPOINT updt_conf2 ;

UPDATE pilote SET plnom='Conficios' WHERE plnom='Conficius';
ROLLBACK TO updt_conf1 ;

UPDATE pilote SET plnom='Conficius' WHERE plnom='Conficies';

UPDATE pilote SET sal=40000.0 WHERE plnom='Conficius';
COMMIT ;

5.2. Gestion des verrous

Une des raisons d'être d'un SGBD est l'accès concurrent aux données par plusieurs utilisateurs.

Aussi, pour assurer l'accès aux données sans risque d'anomalie de lecture ou de mise à jour, Oracle utilise la technique du verrouillage.

Les verrous permettent d'éviter des interactions entre des utilisateurs qui accèderaient à la même ressource.

Les granules de verrouillage sont : la **table** ou la **ligne**. La pose des verrous s'effectuent de deux façons :

- implicitement : c'est le moteur Oracle qui décide de poser un verrou ;

- explicitement : c'est le programmeur qui pose explicitement les verrous.

5.2. Gestion des verrous (suite ...)

Verrous ligne (TX) :

implicitement, un verrou exclusif est posé lors d'un ordre :

- INSERT ou
- UPDATE ou
- DELETE ou
- SELECT ... FOR UPDATE

Les autres utilisateurs ne pourront accéder aux ressources concernées jusqu'à la libération des verrous.

Verrous table (TM) :

Si une transaction pose un verrou ligne exclusif, il est également posé automatiquement un verrou table sur la table concernée.

Un verrou table est posé lors d'un ordre :

- INSERT ou
- UPDATE ou
- DELETE ou
- SELECT ... FOR UPDATE ou
- LOCK TABLE

5.2. Gestion des verrous (suite ...)

Ce verrou table évite la pose de tout verrou exclusif sur la table pour des opérations de DDL (Data Definition Language)

Ainsi il évite qu'un autre utilisateur puisse modifier ou supprimer la table ... lors d'une interrogation, modification ou suppression de lignes.

Les différents types de Verrous :

X : Verrou EXCLUSIF permet aux autres d'interroger une table mais leur interdit les modifications dans la table et la pose d'un verrou S

S : Verrou SHARE favorise la lecture dans une table mais interdit les MISES A JOUR (X, SRX, RX)

RS: Verrou ROW SHARE (SHARE UPDATE) permet l'accès concurrent à une table. Aide à se prémunir d'un verrou X

RX: Verrou ROW EXCLUSIVE se comporte comme RS mais interdit la pose de verrous S, SRX ou X

SRX: Verrou SHARED ROW EXCLUSIVE. Favorise vos MAJ sans risquer S, SRX, RX ou X

Le **tableau ci-dessous indique les différents modes de verrouillages** et les actions autorisées et interdites

Commande SQL	**Mode de verrouillage (niveau table)**	**Actions autorisées**	**Actions interdites**
SELECT ... FROM table	aucun	Toutes On peut poser les verrous suivants : RS row share RX row exclusive S share SRX share row exclusive X exclusive	aucune
- UPDATE table, - INSERT INTO table, - DELETE FROM table, - LOCK TABLE table IN row exclusive mode	Row exclusif (RX)	- SELECT - INSERT - UPDATE - DELETE On peut poser les verrous suivants : RX, RS	Accès en mode exclusif en lecture et écriture: **S, SRX**, X avec les ordres : LOCK TABLE IN - share mode - share row exclusive - exclusive mode
- SELECT ... FOR UPDATE, - LOCK TABLE table IN ROW SHARE MODE	Row Share (RS)	- SELECT - INSERT - UPDATE - DELETE On peut poser les verrous suivants : RX, RS, S, RSX	Accès en mode exclusif en écriture **X** avec l'ordre : LOCK TABLE In exclusive mode
Commande SQL	**Mode de verrouillage (niveau table)**	**Actions autorisées**	**Actions interdites**

Lock table *table* in share mode	Share (S)	- SELECT - SELECT ... FOR UPDATE On peut poser les verrous suivants : RS, S	- INSERT - UPDATE - DELETE Verrous : RX, SRX, **X** avec les ordres : LOCK TABLE IN - share row exclusive mode - exclusive mode - row exclusive mode
lock table table in share row exclusive mode	Share Row Exclusif (SRX)	- SELECT - SELECT FOR UPDATE Verrous autorisés : RS	- INSERT - UPDATE - DELETE Verrous : RX, S, SRX, **X** avec les ordres : LOCK TABLE IN - share mode - share row exclusive mode - exclusive mode - row exclusive mode
lock table table in exclusive mode	Exclusif (X)	- SELECT Aucun verrou n'est autorisé	Tout sauf les requêtes.

Verrous par défaut

Commande SQL	Verrous ligne	Mode de verrouillage de la table
SELECT	--	--
INSERT	oui	RX
UPDATE	oui	RX
DELETE	oui	RX
SELECT ... FOR UPDATE	oui	RS
LOCK table IN **ROW SHARE** MODE	--	RS
LOCK table IN **ROW EXCLUSIVE** MODE	--	RX
LOCK table IN **SHARE EXCLUSIF** MODE	--	SRX
LOCK table IN **SHARE** MODE	--	S
LOCK table IN **EXCLUSIVE** MODE	--	X
DDL / DCL	--	X

5.2. Gestion des verrous (suite ...)

Verrous au niveau table

La clause de verrouillage explicite au niveau d'une table est :

LOCK TABLE <liste_de_table>

IN <type_de_verrou>

MODE [NOWAIT]

(voir le tableau ci-dessus)

Note :

- si un utilisateur A tente de verrouiller une table dans un mode incompatible avec un verrou posé par l'utilisateur B, il est mis en attente jusqu'à ce que B fasse COMMIT ou ROLLBACK (libère les verrous).

L'option NOWAIT permet d'éviter le blocage de A si la ressource n'est pas libre.

Exemple :

T1 (B) : LOCK TABLE pilote EXCLUSIF MODE ;

T2 (A) : LOCK TABLE pilote SHARED UPDATE
NOWAIT ;

5.2. Gestion des verrous (suite ...)

Verrous au niveau ligne

Les commandes SELECT FOR UPDATE, INSERT, DELETE, UPDATE placent un verrou exclusif sur une ou plusieurs lignes d'une table.
Au niveau table, un verrou RS (row share) est posé pour la commande SELECT FOR UPDATE et un verrou RX (row exclusif). Un verrou ligne est toujours posé implicitement.

Les verrous acquis grâce à la commande LOCK TABLE sont des verrous tables, ils ne verrouillent pas directement des lignes mais servent à se prémunir de certains verrous.

Exemple : SELECT * FROM PILOTE
WHERE ADR='Paris'
FOR UPDATE OF SAL ;

Cette commande verrouille les lignes de tous les pilotes habitant Paris. Toutes les lignes sélectionnées sont verrouillées, pas seulement les champs apparaissant après OF.
Ensuite on peut effectuer des mises à jours comme suit :

```
UPDATE PILOTE
SET SAL = 1.1 * SAL
WHERE ADR='Paris';
```

Note : L'option FOR UPDATE ne s'emploie pas avec DISTINCT, GROUP BY, les opérateurs ensemblistes et les fonctions de groupes.

EXERCICES Série 13

Pour effectuer ces tests il est nécessaire d'ouvrir deux sessions.
Commentez les étapes où il y a un ?

Transaction 1	Temps	Transaction 2
LOCK TABLE pilote IN ROW SHARE MODE ;	1	
	2	DROP TABLE pilote ; ?
	3	LOCK TABLE pilote IN EXCLUSIVE MODE NOWAIT ?
	4	select sal from pilote where pilote.adr= 'Paris' FOR UPDATE OF sal ;
UPDATE pilote set sal = 12000.9 where adr = 'Paris'; (attente de T2)	5	
	6	**ROLLBACK ;** (libération des lignes verrouillées par T2)
?	7	
LOCK TABLE pilote IN ROW EXCLUSIVE MODE	8	
	9	LOCK TABLE pilote IN EXCLUSIVE MODE NOWAIT; ?
	10	LOCK TABLE pilote IN SHARE ROW EXCLUSIVE MODE NOWAIT ; ?
	11	LOCK TABLE pilote IN SHARE MODE NOWAIT; ?

	12	UPDATE pilote set sal = 12000.9 where adr = 'Paris'; ?
	13	ROLLBACK; ?
SELECT sal FROM pilote WHERE adr='Paris' FOR UPDATE of sal; ?	14	
	15	UPDATE pilote set sal = 12000.9 where adr = 'Paris'; (attente T1)
ROLLBACK;	16	
	17	? ROLLBACK;
LOCK TABLE pilote IN SHARE MODE ; ?	18	
	19	LOCK TABLE pilote IN EXCLUSIVE MODE NOWAIT ; ?
	20	LOCK TABLE pilote IN SHARE ROW EXCLUSIVE MODE NOWAIT ; ?
	21	LOCK TABLE pilote IN SHARE MODE ; ?
	22	select sal from pilote where user.adr= 'Paris'; ?
	23	select sal from pilote where adr= 'Paris' FOR UPDATE OF sal ; ?
	24	UPDATE pilote set sal = 12000.9 where adr = 'Paris'; (attente T1)
ROLLBACK ;	25	
	26	X lignes modifiées (après libération du verrou par T1) ROLLBACK ;
LOCK TABLE pilote IN SHARE ROW EXCLUSIVE MODE; ?	27	

	28	LOCK TABLE pilote in EXCLUSIVE MODE NOWAIT; ?
	29	LOCK TABLE pilote in SHARE ROW EXCLUSIVE MODE NOWAIT; ?
	30	LOCK TABLE pilote in SHARE MODE NOWAIT; ?
	31	LOCK TABLE pilote in ROW EXCLUSIVE MODE NOWAIT; ?
	32	LOCK TABLE pilote IN ROW SHARE MODE; ?
	33	select sal from pilote where adr= 'Paris'; ?
	34	select sal from pilote where adr= 'Paris' FOR UPDATE OF sal ; ?
	35	UPDATE pilote set sal = 12000.9 where adr = 'Paris'; (attente T1)
UPDATE pilote set sal = 12000.9 where adr = 'Paris'; (attente T2)	36	deadlock
?	37	
	38	? ROLLBACK;

LOCK TABLE pilote IN EXCLUSIVE MODE; ?	39	
	40	LOCK TABLE pilote IN EXCLUSIVE MODE NOWAIT; ?
	41	LOCK TABLE pilote IN SHARE ROW EXCLUSIVE MODE NOWAIT; ?
	42	LOCK TABLE pilote IN SHARE MODE NOWAIT; ?
	43	LOCK TABLE pilote IN ROW EXCLUSIVE MODE NOWAIT; ?
	44	LOCK TABLE pilote IN ROW SHARE MODE NOWAIT; ?
	45	select sal from pilote where adr= 'Paris' ?
	46	select sal from pilote where adr= 'Paris' FOR UPDATE OF sal ; ?
UPDATE pilote set sal = 12000.9 where adr = 'Paris'; ?	47	
COMMIT;	48	
	49	?

5.2. Gestion des verrous (suite ...)

Choix d'un type de verrou

Le moteur Oracle pose des verrous implicites. Ceux-ci peuvent être remis en cause par l'utilisateur grâce aux commandes SELECT FOR UPDATE ou LOCK TABLE. Les verrous explicites pouvant influencer négativement les performances, leurs pose doit se faire avec précautions :

- ne poser de verrous exclusifs sur une table que si le traitement l'exige ;

- poser les verrous de préférence au niveau ligne pour favoriser la concurrence ;

- poser un verrou share (S) pour favoriser les applications en lecture.

Un verrou ligne favorise la concurrence mais pénalise les applications effectuant de nombreuses mises à jours. Dans le cas d'une application bancaire, la réalisation de l'opération DEBIT-CREDIT nécessite probablement un verrou EXCLUSIVE (X) au niveau table. La consultation de la table restant possible, les transactions de consultations de comptes se feront parallèlement.

6. Le Schéma externe (les vues)

Une vue est une table logique qui permet l'accès aux données de une ou plusieurs tables de bases de façon transparente.
Une vue ne contient aucune ligne. Les données sont stockées dans les tables.

Les vues sont utilisées pour :

- assurer l'indépendance logique/externe ;

- fournir un niveau supplémentaire de sécurité sur les tables de base.
Ainsi on peut restreindre, pour un utilisateur donné, l'accès à qq. lignes d'une table ;

- masquer la complexité : une vue peut être la jointure de N-tables ;

- fournir une nouvelle vision de la base. Au lieu de changer le nom des tables de base, on changera seulement au niveau d'une vue si le changement ne concerne pas toute les applications ;

- masquer les bases de données distantes.

6.1. Définition du schéma externe

CREATE [OR REPLACE]
[FORCE | NOFORCE] **VIEW** nom_de_vue
[(alias_colonne1, alias_colonne2, ...)]
AS subquery
WITH CHECK OPTION [CONSTRAINT constraint] ;

Note :
- OR REPLACE : permet de supprimer puis de recréer la vue si elle existe
- FORCE : ignore les erreurs et crée la vue
- WITH CHECK OPTION : permet d'assurer la cohérence des informations modifiées afin de laisser dans la vue les lignes affectées par une modification

Remarques :
- la modification d'une table de base affecte la vue
- le corps d'une vue ne peut contenir de clause ORDER BY ou FOR UPDATE
- on ne peut effectuer des insertions, des mises à jours et des suppressions dans une vue contenant une jointure, des opérateurs ensemblistes, des fonctions de groupe, les clauses GROUP BY, CONNECT BY ou START WITH et l'opérateur DISTINCT.
- tables systèmes: all_views, dba_views, user_views

6.1. Définition du schéma externe (suite ...)

Création d'une vue pour assurer l'intégrité : *contraintes structurelles (ORACLE V6)*

La gestion de l'intégrité de cette façon n'est pas intéressante avec la version 7 d'Oracle. Elle est prise en charge dans le noyau. Dans Oracle 6 l'approche ci-dessous est utile.

Intégrité de domaine

Note : - avec oracle 7 on peut utiliser les clauses CONSTRAINT ou CHECK dans la structure d'une table.
- pas de domaine sémantique

Intégrité de relation (ou d'entité)

Deux conditions pour assurer l'unicité de la clé doivent être remplies :
- pas de valeurs nulles dans la clé (option NOT NULL) ;
- pas de doublon (CREATE UNIQUE INDEX).

Note :
Avec Oracle 7 la clause PRIMARY KEY permet de prendre en compte automatiquement l'intégrité d'entité.

6.1. Définition du schéma externe (suite ...)

Intégrité de référence

La vérification des contraintes d'intégrités de référence sous Oracle V6 peut être simulée grâce aux vues avec la clause WITH CHECK OPTION qui assure la vérification de la contrainte.

Note : sous Oracle 7, les clauses REFERENCES et FOREIGN KEY permettent de prendre en compte automatiquement les contraintes d'intégrité de référence.

6.2. Manipulation sur les vues

Une vue est manipulable, comme une table de base, avec les clauses SQL (SELECT, INSERT, DELETE, UPDATE).

Une vue peut servir à la construction de requêtes imbriquées puisqu'elle peut apparaître derrière la clause FROM.

Opérations autorisées

* Vue avec jointure :
Delete : NON Update : NON Insert : NON

* Vue avec GB ou Distinct :
Delete : NON Update : NON Insert : NON

* Vue avec référence à RowNum :
Delete : NON Update : NON Insert : NON

* Vue avec une colonne issue d'une expression :
Delete : OUI Update : OUI sur autres col. Insert : OUI sur autre col.

* Vue avec au moins une colonne NOT NULL absente :
Delete : OUI Update : OUI Insert : NON

EXERCICES Série 14

Indépendance logique/externe : *vue de sélection*
- "Créer une vue AVA300 qui donne tous les A300 dans la compagnie"
- "Que se passe - t-il à l'insertion d'un "B707" dans la vue ?"

Indépendance logique/externe : *renommage et ré-ordonnancement des colonnes*
- "Créer une vue PAYE qui donne pour chaque pilote son salaire mensuel et annuel"
- "Créer une vue AVPLUS qui donne tous les numéros d'avions conduits par plus d'un pilote."
- "Créer une vue PILPARIS qui donne les noms, les numéros de vols, des pilotes qui assurent au moins un vol au départ de Paris"

Création d'une vue pour assurer la confidentialité
"Créer une vue PILSANS qui donne les renseignements concernant les pilotes, sans le salaire."

Création d'une vue pour assurer l'intégrité : *contraintes structurelles (ORACLE V6)*

Intégrité de domaine

"Créer une vue qui assurer les contraintes de domaine suivantes dans la table AVION :
- AVTYPE {A300, A320, Concorde, B707, Caravelle }
- AV# entre 200 et 500

"Créer une vue PIL25 qui vérifie que chaque pilote inséré a plus de 25 ans."

Intégrité de référence

"Créer les tables PILOTE6, AVION6 et VOL6 (sans les clauses REFERENCES et FOREIGN KEY d'Oracle 7) à partir de PILOTE, AVION, VOL. Créer ensuite une vue VOLSURS vérifiant l'intégrité de référence en insertion dans VOL6. La contrainte à vérifier est : pour tout nouveau vol, le pilote et l'avion doivent exister.
Test :

```
insert into volsurs values(150,1,20,'NICE', 'Paris',1345,1500,'3-MAR-89' );
insert into volsurs values(100,1,1,'NICE', 'Nantes',1345,1500,'4-MAR-89' );
```

"Créer une vue PILOTSUP sur PILOTE6 et VOL6 dans laquelle on accède à tous les pilotes qui ne conduisent aucun vol. La contrainte à vérifier est qu'un Pilote ne peut - être supprimé que s'il ne conduit aucun Avion."
Test :

```
delete from pilotsup where pl# = 1;
delete from pilotsup where pl# = 6;
```

Vues issues d'une table

"Créer une vue AVIONNICE : Ensemble des avions localisés à Nice"

Modification à travers une vue

1) Lister l'extension de la vue AVIONNICE
2) Mise à jour d'un tuple dans cette vue : localiser l'avion de n° 5 à Paris
3) Mise à jour d'un tuple dans cette vue : localiser l'avion n° 7 à Paris
4) Lister la table de base AVION. Que constatez-vous ?

Insertion dans la vue

1) Insérer le tuple (11, 'A300', 220, 'Nice', 'EN service');
2) lister l'extension de la vue AVIONNICE
3) lister la table de base AVION.

Suppression dans la vue

1) Suppression de l'avion N° 11
2) lister l'extension de la vue AVIONNICE
3) lister la table AVION.

Vues issues de plusieurs tables

"Créer une vue AVSERVPARIS : Ensemble des avions en service localisés à Paris"

Modification de la vue

1) lister l'extension de la vue AVSERVPARIS
2) mise à jour d'un tuple de cette vue. Que remarque-t-on ?"

Insertion dans la vue

1) recréez la vue avec jointure
2) insertion d'un tuple dans la vue AVSERVPARIS. Que remarque-t-on?

suppression dans la vue

1) suppression de tous les pilotes de n° inférieur à 7 dans AVSERVPARIS

Vues contenant des colonnes virtuelles

"Reprendre la vue PAYE et lister son contenu"

Modification via la vue

1) Mise à jour d'un tuple dans cette vue : mettre le salaire du pilote 1 à 0
2) lister le contenu de cette vue. Que remarque--on ?

Insertion via la vue

1) insertion d'un tuple dans la vue PAYE . Que remarque-t-on ?

Suppression via la vue

1) suppression de tous les pilotes dont le salaire annuel est supérieur à 180000.

Vues contenant une clause GROUP BY

"Reprenons la vue AVPLUS. Llister cette vue"

"Quels sont le n° d'avions conduits par plus d'un pilote et localisés à Paris ?

PL/SQL

1. INTRODUCTION

SQL :

est un langage ensembliste et non procédural

PL/SQL :

est un langage procédural qui intègre des ordres SQL de gestion de la base de données

Instructions SQL intégrées dans PL/SQL :

- SELECT
- INSERT, UPDATE, DELETE
- COMMIT, ROLLBACK, SAVEPOINT
- TO_CHAR, TO_DATE, UPPER, ...

Instructions spécifiques à PL/SQL :

- définition de variables
- traitements conditionnels
- traitements répétitifs
- traitement des curseurs
- traitement des erreurs

2. Structure d'un bloc PL/SQL

Un bloc PL/SQL est divisé en 3 sections :

DECLARE
Déclaration de variables, constantes,
exceptions, curseurs

BEGIN [nom_du_bloc]
Instructions SQL et PL/SQL

EXCEPTION
Traitement des exceptions
(gestion des erreurs)

END [nom_du_bloc] ;

Remarques :
- les sections DECLARE et EXCEPTION sont facultatives.
- chaque instruction se termine par un ;
- Les commentaires :

```
-- sur une ligne
      ou
/* sur plusieurs
   lignes */
```

2. Structure d'un bloc PL/SQL (suite ...)

Exemple :

```
PROMPT nom du produit              SQL
ACCEPT prod

DECLARE                            PL/SQL
      qte NUMBER(5)

BEGIN
      SELECT quantite INTO qte
      FROM stock
      WHERE produit= '&prod';
      -- on contrôle le stock
      IF qte > 0
      THEN
            UPDATE stock
            SET quantite = quantite - 1
            WHERE produit = '&prod';
            INSERT INTO vente
            VALUES('&prod' || 'VENDU' , SYSDATE);
      ELSE    INSERT INTO commande
            VALUES('&prod' || 'DEMANDE' , SYSDATE);
      END IF;
      COMMIT;
END;
/
```

3. Les variables utilisées dans PL/SQL

3.1. Les différents types de variables locales

Les variables locales se déclarent dans la partie DECLARE du bloc PL/SQL.

Différents types de variables :

* Variables de types ORACLE

* Variables de type BOOLEAN

* Variables faisant référence au disctionnaire de données

Initialisation des variables

Visibilité des variables

3.1.1. Variables de type ORACLE

Syntaxe :

```
nom_var  TYPE_ORACLE;
```

Exemple :

```
DECLARE
    nom          CHAR(20);
    prenom  CHAR(15);
    age        NUMBER(3);
BEGIN
    ...
END;
```

3.1.2. Variables de type BOOLEEN

Syntaxe :

nom_var **BOOLEAN**;

Exemple :

```
DECLARE
    retour          BOOLEAN;
BEGIN
    ...
END;
```

3.1.3. Variables faisant référence au dictionnaire de données

* Variable de même type qu'un attribut d'une table de la base

Syntaxe :

```
nom_var      table.colonne%TYPE;
```

Exemple :

```
DECLARE
    nom      pilote.plnom%TYPE;
BEGIN
    ...
END;
```

3.1.3. Variables faisant référence au dictionnaire de données (suite ...)

- Variable de même structure qu'une ligne d'une table de la base

Syntaxe :

```
nom_var      table%ROWTYPE;
```

Exemple :

```
DECLARE
    ligne     pilote%ROWTYPE;
BEGIN
    ...
END;
```

Remarque :

La structure ligne contient autant de variables que de colonnes de la table. Ces variables portent le même nom et sont de même type que les colonnes de la table.

Pour y accéder :

```
ligne.<nom_col1>
ligne.<nom_col2>
    ...
ligne.<nom_coln>
```

3.1.3. Variables faisant référence au dictionnaire de données (suite ...)

- Variable de même type qu'une autre variable

Syntaxe :

nom_var2 nom_var1**%TYPE**;

Exemple :

```
DECLARE
    ancien_sal      NUMBER(5);
    nouveau_sal     ancien_sal%TYPE;--NUMBER(5);
BEGIN
    ...
END;
```

3.1.4. Initialisation des variables

Avec :

opérateur **:=**

ou

SELECT ... INTO ...

Exemple :

```
DECLARE
    var1    CHAR(10) := 'DUPONT';
    var2    NUMBER(5,2) := 100;
    var3    CHAR(10);
    var4    DATE;
BEGIN
    SELECT col1, col2
    INTO var3, var4
    FROM ... ;
        ...
END;
```

Remarque :
le SELECT doit ramener une et une seule ligne, sinon erreur.

3.1.5. Visibilité des variables

Une variable est visible dans le bloc où elle a été déclarée et dans les blocs imbriqués si elle n'a pas été redéfinie.

```
DECLARE
      var1      NUMBER(3);
      var2      CHAR(10);
BEGIN
      ...                                 var1      NUMBER(3)
      ...                                 var2      CHAR(10)
      DECLARE
            var1      CHAR(10);
            var3      DATE;
      BEGIN
            ...                           var1      CHAR(10)
            ...                           var2
            ...                           var3
      END;
      ...
      DECLARE
            var4      NUMBER(5,2);
      BEGIN
            ...                           var1 NUMBER(3)
            ...                           var2
            ...                           var4
      END;
   ...                                    var1      NUMBER(3)
   ...                                    var2      CHAR(10)
END;
```

3.2. Variables de l'environnement extérieur à PL/SQL

Outre les variables locales vues précédemment, un bloc PL/SQL peut utiliser d'autres variables :

- les champs d'écrans FORMS,
- les variables définies en langage hôte (préfixée de :)
- les variables définies dans SQL*Plus (préfixée de &)

4. Les traitements

4.1. IF : traitement conditionnel

Exécution d'un traitement en fonction d'une condition.

IF condition1 THEN traitement 1;

ELSIF condition2 THEN traitement 2;

[ELSE traitement 3;]

END IF;

Les opérateurs utilisés dans les conditions sont les mêmes que dans SQL :

=, <, ... IS NULL, LIKE, ...

Dès que l'une des conditions est vraie, le traitement qui suit le THEN est exécuté.
Si aucune condition n'est vraie, c'est le traitement qui suit le ELSE qui est exécuté.

4.2. Boucle de base LOOP : traitement répétitif

Exécution d'un traitement plusieurs fois, le nombre n'étant pas connu mais dépendant d'une condition.

```
BEGIN
      LOOP [label]
            instructions;
      END LOOP [label];
END;
```

Pour sortir de la boucle, utiliser la clause :

```
      EXIT [lable] WHEN condition
```

Exemple : insérer les 10 premiers chiffres dans la table result

```
      DECLARE
            nb   NUMBER := 1;
      BEGIN
        LOOP
            INSERT INTO result
            VALUES (nb);
            nb := nb + 1;
            EXIT WHEN nb > 10;
        END LOOP;
      END ;
```

4.3. Boucle FOR : traitement répétitif

Exécution d'un traitement un certain nombre de fois. Le nombre étant connu.

```
BEGIN
      FOR indice IN [REVERSE] exp1 ... exp2
      LOOP
            instructions;
      END LOOP;
END;
```

Remarques :
- inutile de déclarer indice
- indice varie de exp1 à exp2 de 1 en 1
- si REVERSE est précisé, indice varie de exp2 à exp1 avec un pas de -1.

Exemple : calcul de la factorielle 5

```
      DECLARE
            fact NUMBER := 1;
      BEGIN
            FOR i IN 1 .. 5
            LOOP
                  fact := fact * i ;
            END LOOP;
      END;
```

4.4. Boucle WHILE : traitement répétitif

Exécution d'un traitement trant qu'une condition reste vraie.

```
BEGIN

      WHILE condition
      LOOP
            instructions;
      END LOOP;

END;
```

Exemple : reste de la division de 5432 par 5

```
DECLARE
      reste NUMBER := 5432;
BEGIN
      WHILE reste >= 5
      LOOP
            reste := reste -5;
      END LOOP;
END;
```

5. Les curseurs en PL/SQL

5.1. Définitions

Il existe 2 types de curseurs :

- CURSEUR IMPLICITE :

curseur SQL généré et géré par le noyau pour chaque ordre SQL d'un bloc.

- CURSEUR EXPLICITE :

curseur SQL généré et géré par l'utilisateur pour traiter un ordre SELECT qui ramène plus d'une ligne.

5.2. Curseur explicite

4 étapes :

- déclaration du curseur
- ouverture du curseur
- traitement des lignes
- fermeture du curseur

5.2. Curseur explicite (suite ...)

Déclaration du curseur

déclaration dans la section DECLARE du bloc.
on indique le nom du curseur et l'ordre SQL associé

Syntaxe :

CURSOR nom_curseur **IS** ordre_select ;

Exemple :

```
DECLARE
        CURSOR pl_nice IS
            SELECT  pl#, plnom
            FROM    pilote
            WHERE   adr='Nice';
    BEGIN
        ...
    END ;
```

5.2. Curseur explicite (suite ...)

Ouverture du curseur

L'ouverture du curseur lance l'exécution de l'odre SELECT associé au curseur.
Ouverture dans la section BEGIN du bloc.

Syntaxe :

OPEN nom_curseur ;

Exemple :

```
DECLARE
        CURSOR pl_nice IS
            SELECT  pl#, plnom
            FROM    pilote
            WHERE   adr='Nice';
    BEGIN
        ...
        OPEN pl_nice;
        ...
    END ;
```

5.2. Curseur explicite (suite ...)

Traitement des lignes

Après l'exécution du SELECT les lignes ramenées sont traitées une par une, la valeur de chaque colonne du SELECT doit être stockée dans une variable réceptrice.

Syntaxe :

FETCH nom_curseur **INTO** liste_variables ;

Exemple :

```
DECLARE
        CURSOR pl_nice IS
            SELECT  pl#, plnom, sal
            FROM    pilote
            WHERE   adr='Nice';
        num         pilote.pl#%TYPE;
        nom         pilote.plnom%TYPE;
        salaire     pilote.sal%TYPE;
    BEGIN
        OPEN pl_nice;
        LOOP
            FETCH pl_nice INTO num,
                    nom,salaire;
            ...
            EXIT WHEN sal > 10 000;
        END LOOP;
    END ;
```

5.2. Curseur explicite (suite ...)

Fermeture du curseur

Pour libérer la mémoire prise par le curseur, il faut le fermer dès qu'on n'en a plus besoin.

Syntaxe :

CLOSE nom_curseur ;

Exemple :

```
DECLARE
        CURSOR pl_nice IS
            SELECT  pl#, plnom, sal
            FROM    pilote
            WHERE   adr='Nice';
        num         pilote.pl#%TYPE;
        nom         pilote.plnom%TYPE;
        salaire     pilote.sal%TYPE;
    BEGIN
        OPEN pl_nice;
        LOOP
            FETCH pl_nice INTO num, nom,salaire;
            ...
            EXIT WHEN sal > 10 000;
        END LOOP;
        CLOSE pl_nice;
    END ;
```

5.3. Les attributs d'un curseur

Pour tout curseur (implice ou explicite) il existe des indicateurs sur leur état.

%FOUND	dernière ligne traitée
%NOTFOUND	
%ISOPEN	ouverture d'un curseur
%ROWCOUNT	nombre de lignes déjà traitées

5.3.1. %FOUND

curseur implicite : SQL%FOUND

TRUE

* si INSERT, UPDATE, DELETE traite au moins une ligne

* si SELECT ... INTO ... ramène une et une seule ligne

curseur explicite : nom_curseur%FOUND

TRUE

* si le dernier FETCH a ramené une ligne.

5.3.1. %FOUND (suite ...)

Exemple :

```
DECLARE
    CURSOR pl_nice IS
        SELECT  pl#, plnom, sal
        FROM    pilote
        WHERE   adr='Nice';
    num         pilote.pl#%TYPE;
    nom         pilote.plnom%TYPE;
    salaire     pilote.sal%TYPE;

BEGIN
    OPEN pl_nice;
    FETCH pl_nice INTO num, nom,salaire;
    WHILE pl_nice%FOUND
    LOOP
        ...
        FETCH pl_nice INTO num,
                nom,salaire;
    END LOOP;
    CLOSE pl_nice;
END ;
```

5.3.2. %NOTFOUND

curseur implicite : SQL%NOTFOUND

TRUE

* si INSERT, UPDATE, DELETE ne traite aucune ligne

* si SELECT ... INTO ... ne ramène pas de ligne

curseur explicite : nom_curseur%NOTFOUND

TRUE

* si le dernier FETCH n'a pas ramené de ligne.

5.3.3. %ISOPEN

curseur implicite : SQL%ISOPEN

toujours à FALSE car ORACLE referme les curseurs après utilisation.

curseur explicite : nom_curseur%ISOPEN

TRUE si le curseur est ouvert.

Exemple :

```
DECLARE
        CURSOR pl_nice IS
            SELECT  pl#, plnom, sal
            FROM    pilote
            WHERE   adr='Nice';
        num         pilote.pl#%TYPE;
        nom         pilote.plnom%TYPE;
        salaire     pilote.sal%TYPE;
    BEGIN
        IF NOT(pl_nice%ISOPEN)
        THEN
            OPEN pl_nice;
        END IF;
            ...
    END ;
```

5.3.4. %ROWCOUNT

curseur implicite : SQL%ROWCOUNT

nombre de lignes traitées par INSERT, UPDATE, DELETE

0 : SELECT ... INTO : ne ramène aucune ligne
1 : SELECT ... INTO : ramène 1 ligne
2 : SELECT ... INTO : ramène plus d'une ligne

curseur explicite : nom_curseur%ROWCOUNT

traduit la nième ligne ramenée par le FETCH

5.4. Simplification d'écriture

5.4.1. Déclaration de variables

Au lieu de déclarer autant de variables que d'attributs ramenés par le SELECT du curseur, on peut utiliser une structure.

Syntaxe :

```
DECLARE
    CURSOR nom_curseur IS ordre_select;
    nom_structure nom_curseur%ROWTYPE;
```

Pour renseigner la structure :

```
FETCH nom_curseur INTO nom_structure;
```

Pour accéder aux éléments de la structure :

```
nom_structure.nom_colonne
```

5.4.2. Traitement du curseur

Au lieu d'écrire :

```
DECLARE
      CURSOR nom_curseur IS SELECT ... ;
      nom_struct  nom_curseur%ROWTYPE;
BEGIN
      OPEN nom_curseur;
      LOOP
            FETCH nom_curseur INTO nom_struct;
            EXIT WHEN nom_curseur%NOTFOUND;
            ...
      END LOOP;
      CLOSE nom_curseur;
END;
```

il suffit d'écrire :

```
DECLARE
      CURSOR nom_curseur IS SELECT ... ;
BEGIN
      FOR nom_struct IN nom_curseur LOOP
            ...
      END LOOP;
END;
```

5.4.2. Traitement du curseur (suite ...)

ou encore :

```
FOR nom_struct IN (SELECT ...)
    LOOP
        ...
    END LOOP;
```

6. Gestion des erreurs en PL/SQL

La section EXCEPTION permet de gérer les erreurs survenues lors de l'exécution d'un bloc PL/SQL.

2 types d'erreurs :

- erreur ORACLE

- erreur utilisateur

6. Gestion des erreurs en PL/SQL(suite)

Syntaxe erreur utilisateur :

```
DECLARE
  nom_erreur EXCEPTION;     on donne un nom à l'erreur
            ...
BEGIN
  IF ...
  THEN RAISE nom_erreur;          on déclenche l'erreur
            ...
EXCEPTION
      WHEN nom_erreur THEN ...  traitement de l'erreur
END;
```

Remarque : on sort du bloc après le traitement de l'erreur.

6. Gestion des erreurs en PL/SQL(suite)

Syntaxe erreur ORACLE non prédéfinie :

```
DECLARE
  nom_erreur EXCEPTION;  on donne un nom à l'erreur
  PRAGMA EXCEPTION_INIT(nom_erreur,code_erreur)
          on associe le nom de l'erreur à un code erreur

          ...
BEGIN
          ...       l'erreur Oracle est détectée par le système
EXCEPTION
      WHEN nom_erreur THEN ...  traitement de l'erreur
END;
```

Remarque :
on sort du bloc après le traitement de l'erreur.

6. Gestion des erreurs en PL/SQL(suite)

Syntaxe erreur ORACLE prédéfinie :

Certaines erreurs ORACLE ont déjà un nom. Il est donc inutile de les déclarer comme précédemment. On utilise leur nom dans la section EXCEPTION.

```
DECLARE
        ...
BEGIN
        ...      l'erreur Oracle est détectée par le système
EXCEPTION
    WHEN nom_erreur THEN ...  traitement de l'erreur
END;
```

Exemple d'erreurs prédéfinies :
- DUP_VAL_ON_INDEX
- NO_DATA_FOUND
- ...
- OTHERS

6. Gestion des erreurs en PL/SQL(suite)

Complément

- SQLCODE renvoie le code de l'erreur courante (numérique)

- SQLERRM[(code_erreur)] renvoie le libellé de l'erreur courante ou le libellé de l'erreur dont le numéro est passé en paramètre.

7. Exercices PL/SQL

7.1. Ex1 : les boucles

Créer une table MULTIPLICATION(op CHAR(5), res char(3)).

Ecrire un fichier de commande qui permette le calcul et l'affichage d'une table de multiplication.
Résoudre l'exerice de 3 manières différentes (utiliser les 3 boucles)

7.2. Ex2 : les curseurs

Ecrire un fichier qui recherche le nième et n+1ème pilote plus agé (recherche sur la date de naissance)

7.3. Ex3 : les erreurs

Ecrire un fichier qui mette à jour, l'âge des pilotes de la table pilote.

Traiter les anomalies :
- pilote de - de 20 ans
- pour les autres erreurs qui pourraient se produire : les traiter globalement.

SQL*Plus

1. Présentation de SQL*Plus

SQL*Plus est un outil Oracle permettant l'accés aux données des bases de données Oracle.

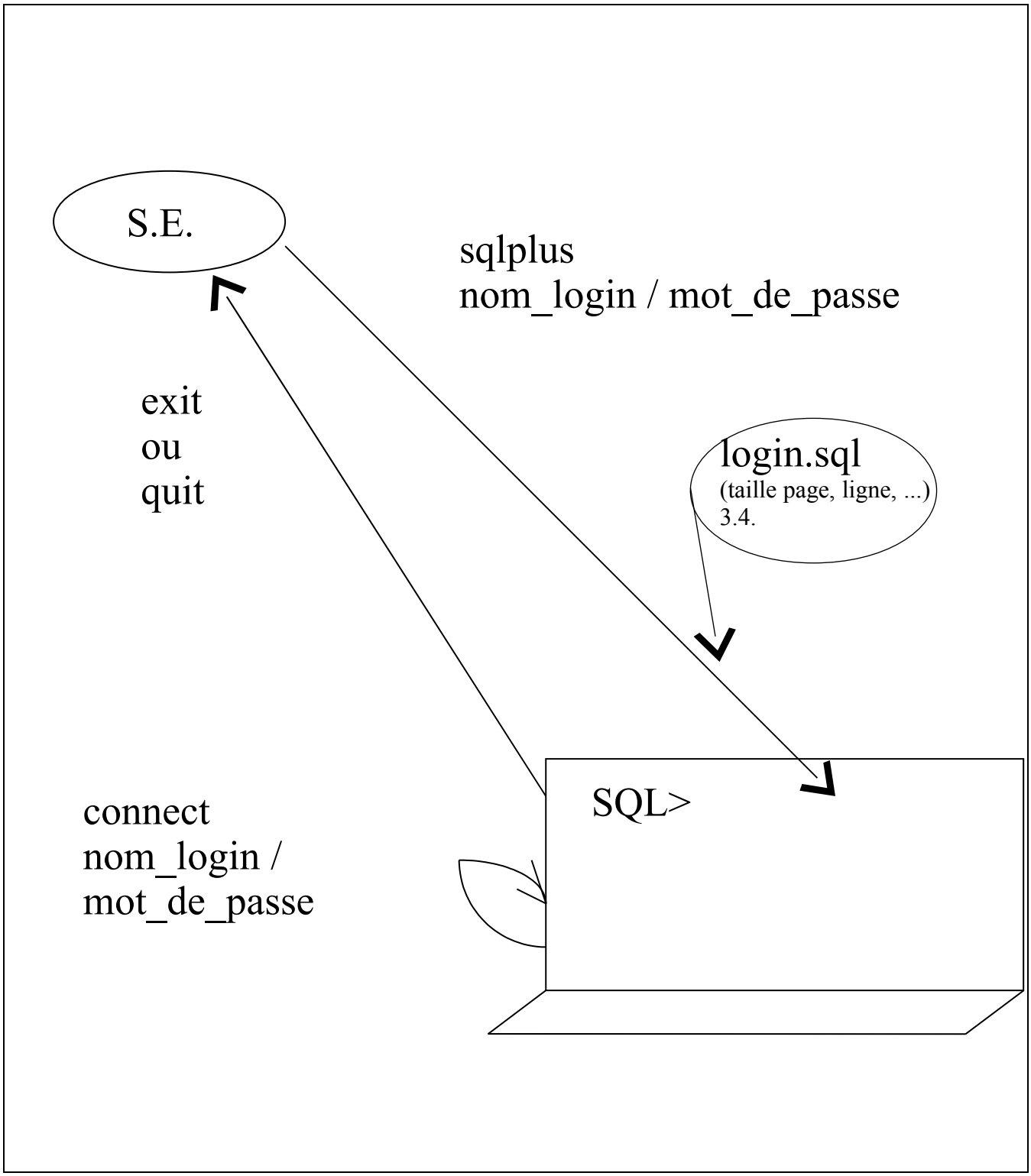
Il autorise les commandes suivantes :

⇨ ordre SQL

⇨ commandes liées à l'éditeur

⇨ commandes liées à l'environnement

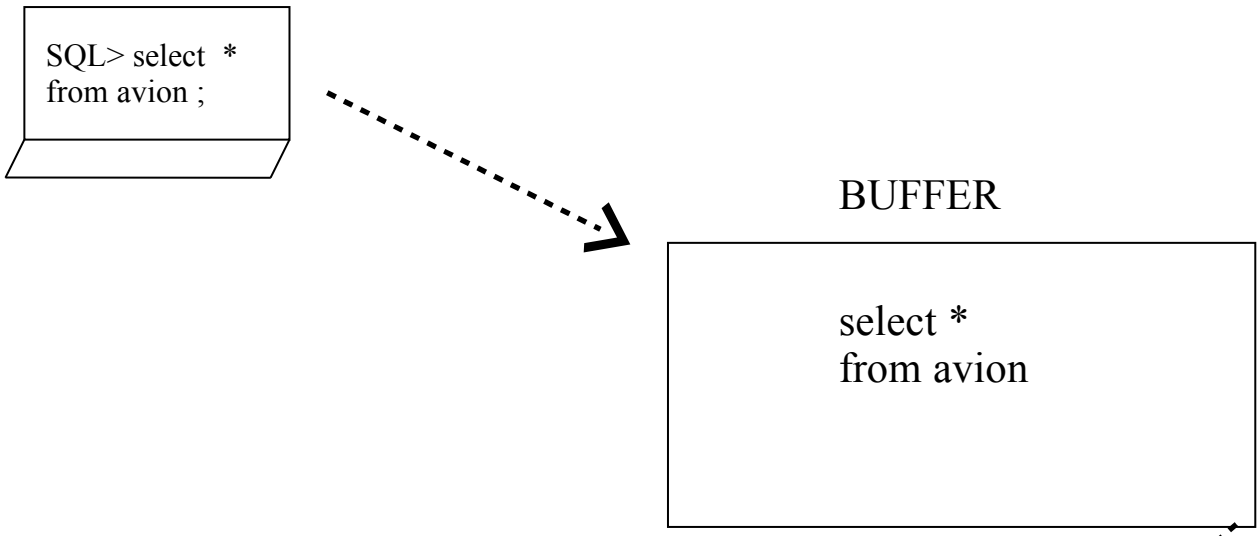
1. Présentation de SQL*Plus (suite)



1. Présentation de SQL*Plus (suite)

Toute commande SQL est mise dans un buffer.

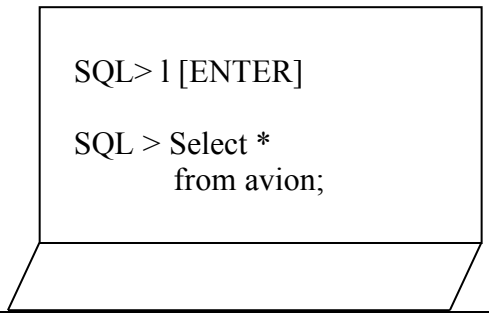
Pour exécuter la commande, on la termine par un **;** ou un / à la ligne suivante.



L'utilisateur aura accés à ce buffer via les commandes de l'éditeur.

Note : pour exécuter de nouveau la commande du buffer, utiliser la commande **R** ou /.

Exemple :



2. Les commandes de l'éditeur

L	Liste le contenu du buffer
L*	Liste la ligne courante
Ln	Liste la nième ligne qui devient courante
Lm n	Liste de la ligne m à la ligne n
i	Insertion après la ligne courante
i texte	Insertion d'une ligne dont le contenu est texte
a	Ajoute du texte en fin de ligne courante
del	Supprime la ligne courante
C/ch1/ch2	Remplace la première occurrence de la chaîne 1 par la chaîne 2 dans la ligne courante
C/ch	Supprime la chaîne dans la ligne courante
clear buffer	Efface le buffer

3. Les commandes de l'environnement

Différents types de commandes d'environnement :

⇨ commandes d'E/S

⇨ commandes de dialogue

⇨ commandes de formatage de rapports et d'édition

⇨ commandes de définition de l'environnement

3. Les commandes de l'environnement (suite)

3.1. Commandes d'Entrées / Sorties

Pour :
⇨ stocker des requêtes dans un fichier
⇨ récupérer des requêtes se trouvant dans un fichier
⇨ stocker des résultats

Les commandes d'E/S :

ed nom_file	ouvre un fichier nom_file.sql sous l'éditeur associé
save nom_file	sauve le contenu du buffer dans nom_file.sql
get nom_file	charge le fichier nom_file.sql dans le buffer
start nom_file ou @nom_file	ouvre le fichier nom_file.sql, le charge dans le buffer et exécute les commandes.
spool nom_file	ouvre un fichier d'impression (.lis ou .lst) qui contiendra la trace écran
spool off	ferme le fichier d'impression
spool out	id. plus envoi à l'imprimante

3.2. Commandes de dialogue

Les commandes de dialogue vont permettre de rendre les procédures paramétrées plus conviviales.

Mais avant de voir ces commandes de dialogue voyons la notion de variables qui permettent le paramétrage des procédures.

Les variables :

Les variables sont définies :

- lors de la connexion (login.sql)

- lors de la session :

    - pour la durée de la session

    - pour une exécution

Exemple de requête paramétrée :

```
SQL> SELECT *
        FROM avion
        WHERE av# = &num_avion
        /

ENTER VALUE FOR num_avion : <saisie du user>

-- exécution de la requête avec la valeur saisie --
```

Remarque :
&var : la variable est définie pour une exécution
&&var : la variable est définie pour la session

Complément :
L'utilisateur pourra donner les valeurs lors de l'appel du script contenant les commandes SQL. Pour cela les variables devront s'appeler &1, &2, ... &9.

Exemple : fichier test.sql

```
Select * from &1 where av# = &2;
```

lancement :

```
SQL> @test avion 20
```

DEF	renvoie les valeurs de toutes les variables définies
DEF var = valeur	affecte une valeur à la variable de façon permanente
UNDEF var	supprime la définition de la variable

Les commandes de dialogue

Rappel :

```
SQL> SELECT *
        FROM avion
        WHERE av# = &num_avion
        /

ENTER VALUE FOR num_avion : <saisie du user>
```

Les commandes de dialogue vont permettre de donner la main à l'utilisateur de façon plus conviviale.

prompt texte — affiche le message à l'utilisateur

accept var [PROMPT texte]
— attend que l'utilisateur rentre une une valeur pour var

Exemple : test2.sql

```
PROMPT entrez le numéro de l'avion
ACCEPT num_avion PROMPT 'Num : '
SELECT *
FROM avion
WHERE av# = &num_avion;
```

3.3. Commandes de formatage de rapport

Les commandes de formatage de rapport et d'édition vont permettre :

- la définition des titres

- le changment des noms et de la taille des colonnes

- la mise en place de rupture pour réaliser des opérations

⇨ COL(umn) nom_col

[FORMAT format] *def. la taille et le format*
ex : A10 (alpha), 99990 (num.)
[HEADING texte] *en_tête de la col.*
[JUSTIFY L|C|R] *cadre l'en-tête de la col.*
[NEW_VALUE var] *met la nvelle val. dans var*
[OLD_VALUE var] *met l'ancienne val. dans var*
[PRINT |NOPRINT] *affiche ou pas la col.*
[NEWLINE] *passage à la ligne*

Exemple : COL pilnom FORMAT A16
HEADING NOM DU | PILOTE

Remarques :
COL renvoie toutes les informations de toutes les colonnes.
COL nom_col id. pour une colonne
COL nom_col CLEAR supprime le «formatage» de la col.
CLEAR COL Id. pour toutes les colonnes.

⇨ **TTITLE texte**

texte est le titre qui apparaîtra à chaque page du rapport

Exemple : TTITLE « AVIONS DE LA COMPAGNIE »

| : pour un saut à la ligne
- : suite ligne suivante

Remarque : TTITLE sans option génère :
- la date en haut à gauche
- le n° de page en haut à droite
- le titre centré
(nom valable pour TTILE avec options)

TTITLE [OPTIONS [texte|var]] | [ON | OFF]
Options : LEFT, CENTER, RIGHT, BOLD, ...
Variables :
SQL.LNO : n° de ligne
SQL.PNO : n° de page
SQL.RELEASE : version d'ORACLE
SQL.USER : nom utilisateur

⇨ **BTITLE texte**

Id. à TTILE mais en titre de bas de page

TTITLE : renvoie les informations sur le titre haut de page

BTITLE : renvoie les informations sur le titre bas de page

TTITLE ON |OFF : active ou désactive l'emploi d'un titre haut de page

BTITLE ON |OFF : active ou désactive l'emploi d'un titre bas de page

⇨ **BREAK ON critere_de rupture action**

définit une rupture et déclenche une action chaque fois qu'elle se produit

avec :

critere_de_rupture : nom_col | ROW | REPORT

action : SKIP (n|PAGE) DUP|NODUP

exemple : BREAK ON PL# SKIP 2
==> saute deux lignes à chaque changement de pilote.

Remarque : bien entendu les données ramenées par le SELECT doivent être en cohérence avec le BREAK.

BREAK : donne les informations sur les ruptures définies
CLEAR BREAK : annule la rupture définie

⇨ COMPUTE {AVG | COUNT | MAX | MIN | NUMBER | STD | SUM | VAR }
OF col1 {col2 ...}
ON {col_break | ROW | REPORT }

Exemple :
COMPUTE SUM
OF sal
ON adr REPORT

REMARQUE IMPORTANTE :
définir le COMPUTE en cohérence avec le BREAK

COMPUTE renvoie les informations sur le calcul

CLEAR COMPUTE annule les calculs définis

Exemple :

BREAK ON avtype SKIP 1 ON REPORT
COMPUTE SUM OF cap ON avtype REPORT
select * from avion;
(somme des cap. par avtype puis pour tous les enregts.)

3.4. Commandes de définition de l'environnement

L'objectif est d'affecter des valeurs à des variables pour définir des caractéristiques :

- d'impression

- d'affichage, d'exécution

SET nom_variable valeur

affecter une valeur à une variable

SHOW nom_variable

consulter la valeur d'une variable

SHOW ALL

id. pour toutes les variables

⇨ impression

SET LINE	n	nb de car. / ligne
SET PAGES	n	nb lignes / page
SET NUM	n	largeur par def. des nb
SET NEWP	n	nb de lignes à sauter en haut de chaque page
SET NULL	ch	valeur de la chaîne NULL
SET SPACE	n	nb expaces entre les colonnes
SET UND	car	caractère de soulignement des en-têtes de colonnes

⇨ affichage, exécution

SET TERM	ON|OFF	affichage O/N
SET ECHO	ON|OFF	réaffichage de la cde SQL
SET FEED	ON|OFF	nb lignes ramenées
SET SHOW	ON|OFF	affichage des paramètres sql*plus
SET HEA	ON|OFF	affichage du titre des col.
SET PAUSE	ON|OFF|TEXTE	arrêt en fin de page (message)
SET SQLN	ON|OFF	numérotation des lignes du buffer
SET SQLP	texte	prompt de SQL*Plus

SET TIME	ON|OFF	affichage de l'heure à gauche du prompt
SET VERIFY	ON|OFF	affichage des cdes avant et après substitution
SET TIMING	ON|OFF	information sur les temps d'exécution

⇨ **divers**

SET DEFINE car
caractère de substitution pour les paramètres
(& par défaut)

SET SCAN ON|OFF
Autorisation d'utilisation des paramètres

SET SQLTERMINATOR car
Défaut : ;

SET HEADS car|ON|OFF
Défaut : |

SET SUFFIX chaîne extension des fichiers

SET AUTOCOMMIT ON|OFF
pour mettre en place un commit automatique

4. Exercice

1. Faire une interrogation de tous les attibuts et de toutes les lignes de la relation AVION.
Résultat trié par type d'avion.
Remarquez la présentation des réultats peu présentable.

2. Faites un describe de la relation AVION.
Remarquez que AV# et LOC sont des NUMBER(4). Pourtant dans le SELECT précédent leur affichage était sur une longueur supérieure. Comment expliquez vous cela ?

3. Modifier la présentation des colonnes avec COL pour que l'interrogation faite en 1. soit plus présentable :
- AV# :
    - titre : N°AVION
    - lg : 10
- AVTYPE :
    - titre TYPE
- CAP :
    - titre : CAPACITE
- LOC :
    - titre : LOCALISATION
- REMARQ :
    - titre : REMARQUE
    - lg : 20

Exécuter la requête.
Corriger la longueur de LOCALISATION pour voir le titre totalement.

4. Titre : Liste des avions de la compagnie
Exécuter la requête.

5. Définir un break sur le type d'avion et sauter 2 lignes.
Exécuter la requête.

6. Caculer la capacité moyenne de tous les avions de la compagnie pour chaque type d'avion
Exécuter la requête.

Annexes

Annexe A :
Schéma et extension de la base aérienne

(fichier airbase1.sql)

```
REM Creation de la base de donnees aerienne


drop table pilote CASCADE CONSTRAINTS;

create table pilote(
      pl#         number(4)              not null primary key,
      plnom       varchar2(12)           not null unique,
      dnaiss      date                   not null,
      adr         varchar2(20)           default 'PARIS',
      tel         varchar2(12),
      sal         number(7,2)            not null check (sal < 70000.0),
      age          number (3)
                  CONSTRAINT pil_chk_age CHECK (age BETWEEN 25 and 60)
);



drop table avion CASCADE CONSTRAINTS ;

create table avion(
      av#         number(4)              not null primary key,
      avtype      varchar2(10)           not null
                  CONSTRAINT avion_chk_type
                  CHECK (avtype in
                        ('A300','A310','A320','B707','B727','CONCORDE','CARAVELLE')),
      cap         number(4)              not null,
      loc         varchar2(20)           not null,
      remarq      long
);
```

```
drop table vol CASCADE CONSTRAINTS ;

create table vol(
      vol#        number(4)      not null primary key,
      pilote#     number(4)      not null
                  CONSTRAINT vol_fk_pilote REFERENCES PILOTE(PL#)
                  ON DELETE CASCADE,
      avion#      number(4)      not null,
      vd          varchar2(20),
      va          varchar2(20),
      hd          number(4)      not null,
      ha          number(4),
      dat         date,
         CONSTRAINT vol_chk_ha CHECK (ha>hd),
         FOREIGN KEY (avion#) REFERENCES AVION(AV#)
);


REM insertion des valeurs dans les tables

insert into  pilote values(1, 'Miranda', '16-AUG-52','Sophia-Antipolis', '93548254', 18009.0,50);
insert into  pilote values(2, 'St-exupery', '16-OCT-32', 'Lyon', '91548254', 12300.0,50);
insert into  pilote values(3, 'Armstrong ', '11-MAR-30', 'Wapakoneta','96548254', 24500.0,50);
insert into  pilote values(4, 'Tintin', '01-AUG-29', 'Bruxelles','93548254', 21100.0,50);
insert into  pilote values(5, 'Gagarine', '12-AUG-34', 'Klouchino','93548454', 22100.0,50);
insert into  pilote values(6, 'Baudry', '31-AUG-59', 'Toulouse','93548444', 21000.0,50);
insert into  pilote values(8, 'Bush', '28-FEB-24', 'Milton','44556254', 22000.0,50);
insert into  pilote values(9, 'Ruskoi', '16-AUG-30', 'Moscou','73548254', 22000.0,50);
insert into  pilote values(10, 'Math', '12-AUG-38', 'Paris', '23548254', 15000.0,50);
insert into  pilote values(11, 'Yen', '19-SEP-42', 'Munich','13548254', 29000.0,50);
insert into  pilote values(12, 'Icare', '17-DEC-62', 'Ithaques','73548211', 17000.6,50);
insert into  pilote values(13, 'Mopolo', '04-NOV-55', 'Nice','93958211', 17000.6,50);
insert into  pilote values(14, 'Chretien', '04-NOV-45', '','73223322', 15000.6,50);
insert into  pilote values(15, 'Vernes', '04-NOV-35', 'Paris', '',17000.6,50);
insert into  pilote values(16, 'Tournesol', '04-AUG-29', 'Bruxelles','', 15000.6,50);
insert into  pilote values(17, 'Concorde', '04-AUG-66', 'Paris', '',21000.6,50);


REM Insertion des avions

insert into  avion values(1, 'A300', 300, 'Nice', 'En service');
insert into  avion values(2, 'A300', 300, 'Nice', 'En service');
insert into  avion values(3, 'A320', 320, 'Paris', 'En service');
insert into  avion values(4, 'A300', 300, 'Paris', 'En service');
insert into  avion values(5, 'CONCORDE', 300, 'Nice', 'En service');
insert into  avion values(6, 'B707', 400, 'Paris', 'En panne');
insert into  avion values(7, 'CARAVELLE', 300, 'Paris', 'En service');
insert into  avion values(8, 'B727', 250, 'Toulouse', 'En service');
insert into  avion values(9, 'CONCORDE', 350, 'Toulouse', 'En service');
insert into  avion values(10, 'A300', 400, 'Paris', 'En service');
REM Insertion des vols

insert into  vol values(100, 1,1,'Nice', 'Paris', '1345', '1500','3-MAR-89' );
```

```
insert into  vol values(110, 3,6,'Nice', 'Toulouse', '1230', '1325','6-MAR-89' );
insert into  vol values(120,4,3,'Nice', 'Paris', '0745', '0900','21-JUN-89' );
insert into  vol values(125, 12,6,'Paris', 'Nice', '1330', '1845','10-JAN-89' );
insert into  vol values(130, 4,8,'Toulouse', 'Beauvais', '0630','0750', '27-MAR-89' );
insert into  vol values(111, 5,3,'Nice', 'Paris', '0800', '0920','4-DEC-89' );
insert into  vol values(135, 8,5,'Paris', 'Toulouse', '1200','1320','22-MAR-89' );
insert into  vol values(140, 14,9,'Lyon', 'Nice', '0700', '0750','4-JUN-89' );
insert into  vol values(150, 1,1,'Paris', 'Nantes','1630', '1725','28-MAR-89' );
insert into  vol values(153, 2,3,'Lyon', 'Nice', '1210', '1300','6-NOV-89' );
insert into  vol values(156, 9,2,'Paris', 'Lyon', '0230', '0320','14-JAN-89' );
insert into  vol values(200, 5,3,'Nice', 'Toulouse', '2030', '2125','17-JUN-89' );
insert into  vol values(210, 14,7,'Nice', 'Nantes', '1430', '1525','14-OCT-89' );
insert into  vol values(236, 8,4,'Lyon', 'Toulouse', '2130', '2250','15-OCT-89' );
insert into  vol values(240, 13,10, 'Nice', 'Paris', '2300', '2355','19-NOV-89' );
insert into  vol values(250, 13,4,'Bordeaux', 'Paris', '2300','2355', '25-DEC-89' );
insert into  vol values(260, 13,5,'Bordeaux', 'Paris', '2300','2355', '30-NOV-89' );
insert into  vol values(270, 13,9,'Paris', 'New york', '1400','2300', '3-JAN-89' );
insert into  vol values(280, 8,9,'Nice', 'Mulhouse', '1200','1320','21-MAR-89' );
insert into  vol values(290, 3,8,'Beauvais', 'Marseille', '1230','1425', '9-MAR-89' );


commit;
```

ANNEXE B : VARIABLES D'ENVIRONNEMENT IMPORTANTES SOUS UNIX

$ echo $ORACLE_HOME #Localisation des fichiers Oracles

$ echo $ORACLE_SID #Nom de l'instance base de données courante