Burkina Faso

Unité- Progrès-Justice

ORACLE DATABASE : SQL/SQL*PLUS ➔ 9 h
ORACLE DEVELOPER : PL/SQL ➔9 h
Devoir 1 : ➔ 1 h 30 min
ORACLE DEVELOPER : FORMS & REPORTS ➔ 9 h
Devoir 2 : ➔ 1 h 30 min

Durée : 30 Heures

Enseignante :
Madame IMA RASMATA
Email : ima_ramatou@yahoo.fr
Tel : 00226 70 73 38 41

Période :

Avril - Mai 2021

Burkina Faso

Unité- Progrès-Justice

ORACLE DATABASE : ORACLE SQL et l'environnement SQL*plus

Durée : 9 Heures

Enseignante :

Madame IMA RASMATA

Période :

Avril - Mai 2021

PROGRAMME

1. INTRODUCTION

- 1.1. DÉFINITIONS
- 1.2. L'OFFRE ORACLE
- 1.3. LES COMMANDES
- 1.4. LES OBJETS

2. INTERROGATION DES DONNÉES

- 2.1. SYNTAXE DU VERBE SELECT
- 2.2. INDÉPENDANCE LOGIQUE EXTERNE
- 2.3. ELIMINATION DE DOUBLONS : DISTINCT
- 2.4. OPÉRATION DE SÉLECTION
  - 2.4.1. Opérateurs arithmétiques
  - 2.4.2. Critères de comparaison : opérateurs sur les chaînes : LIKE
  - 2.4.4. Critères de comparaison avec l'opérateur BETWEEN
  - 2.4.5. Critères de comparaison avec une valeur nulle
  - 2.4.6. Les opérateurs ANY, SOME et ALL

PROGRAMME

2.5. EXPRESSIONS ET FONCTIONS

2.5.1. Les expressions

2.5.2. Les fonctions

2.6. LES FONCTIONS DE GROUPE / UTILISATION DE FONCTIONS AGGRÉGATIVES

2.7. PRÉSENTATION DU RÉSULTAT TRIÉ SELON UN ORDRE PRÉCIS

2.8. REQUÊTES MULTI-RELATIONS SANS SOUS-REQUÊTES : LA JOINTURE OU PRODUIT CARTÉSIEN

2.9. REQUÊTES MULTI-RELATIONS AVEC LES OPÉRATEURS ENSEMBLISTES

2.10. REQUÊTES MULTI-RELATIONS AVEC LES OPÉRATEURS ENSEMBLISTES

2.11. SOUS-INTERROGATIONS NON SYNCHRONISÉE

2.12. LA JOINTURE EXTERNE

2.13. LE PARTITIONNEMENT

PROGRAMME

3. MISE À JOUR DES DONNÉES

- 3.1. INSERTION DE LIGNES
- 3.2. MODIFICATION DE LIGNES
- 3.3. SUPPRESSION DE LIGNES
  - 3.3.1. VIA LA COMMANDE DELETE
  - 3.3.2. VIA LA COMMANDE TRUNCATE

4. LE SCHÉMA DE DONNÉES

- 4.1. DÉFINITION DU SCHÉMA : ASPECTS STATIQUES
  - 4.1.1. LES TYPES DE DONNÉES ORACLE
  - 4.1.2. CRÉATION D'UNE TABLE
  - 4.1.3. CRÉATION D'UN INDEX

PROGRAMME

PROGRAMME

7.GESTION UTILISATEURS, DROITS D'ACCÈS ET OBJETS DE SCHÉMA

(contrôle d'acces des utilisateurs)

7.1. Les privilèges système

- 7.1.1. Qu'est ce qu'un privilège
- 7.1.2. Les privilèges DBA
- 7.1.3. Créer un utilisateur
- 7.1.4. Les privilèges système accordés à un utilisateur
- 7.1.5. Accorder un privilège
- 7.1.5. Accorder un privilège
- 7.1.6 Créer et accorder un privilège à un rôle
- 7.1.6 Créer et accorder un privilège à un rôle
- 7.1.7. Modification de mot de passe

PROGRAMME

7.GESTION UTILISATEURS, DROITS D'ACCÈS ET OBJETS DE SCHÉMA

(contrôle d'acces des utilisateurs)

7.2 Les privilèges objet

7.2.1. Les privilèges objet

7.2.2. Accorder les privilèges

7.2.3. Retirer des privilège

PROGRAMME

8. L'environnement SQL*PLUS

8.1 Les variables de substitution

8.1.1 UTILISATION D'ESPERLUETTE &

8.1.2 SUBSTITUTION DE CHAÎNES DE CARACTÈRES ET DE DATES

8.1.3 UTILISATION DE DOUBLE ESPERLUETTE &&

8.2 Définition des variables de substitution

8.2.1 UTILISATION DE LA COMMANDE DEFINE

8.2.2 UTILISATION DE LA COMMANDE UNDEFINE

8.2.3 UTILISATION DE LA COMMANDE VERIFY

PROGRAMME

7. L'environnement SQL*PLUS

8.3 Personnalisation de l'environnement SQL*PLUS

8.3.1 UTILISATION DE LA COMMANDE SET

8.3.2 LES COMMANDES DE FORMATAGE

8.3.3 UTILISATION DE LA COMMANDE COLUMN

8.3.4 UTILISATION DE LA COMMANDE BREAK

8.3.5 UTILISATION DES COMMANDES TTITLE ET BTITLE

8.3.6 EXÉCUTION DES RAPPORTS FORMATÉS

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 (Structured 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

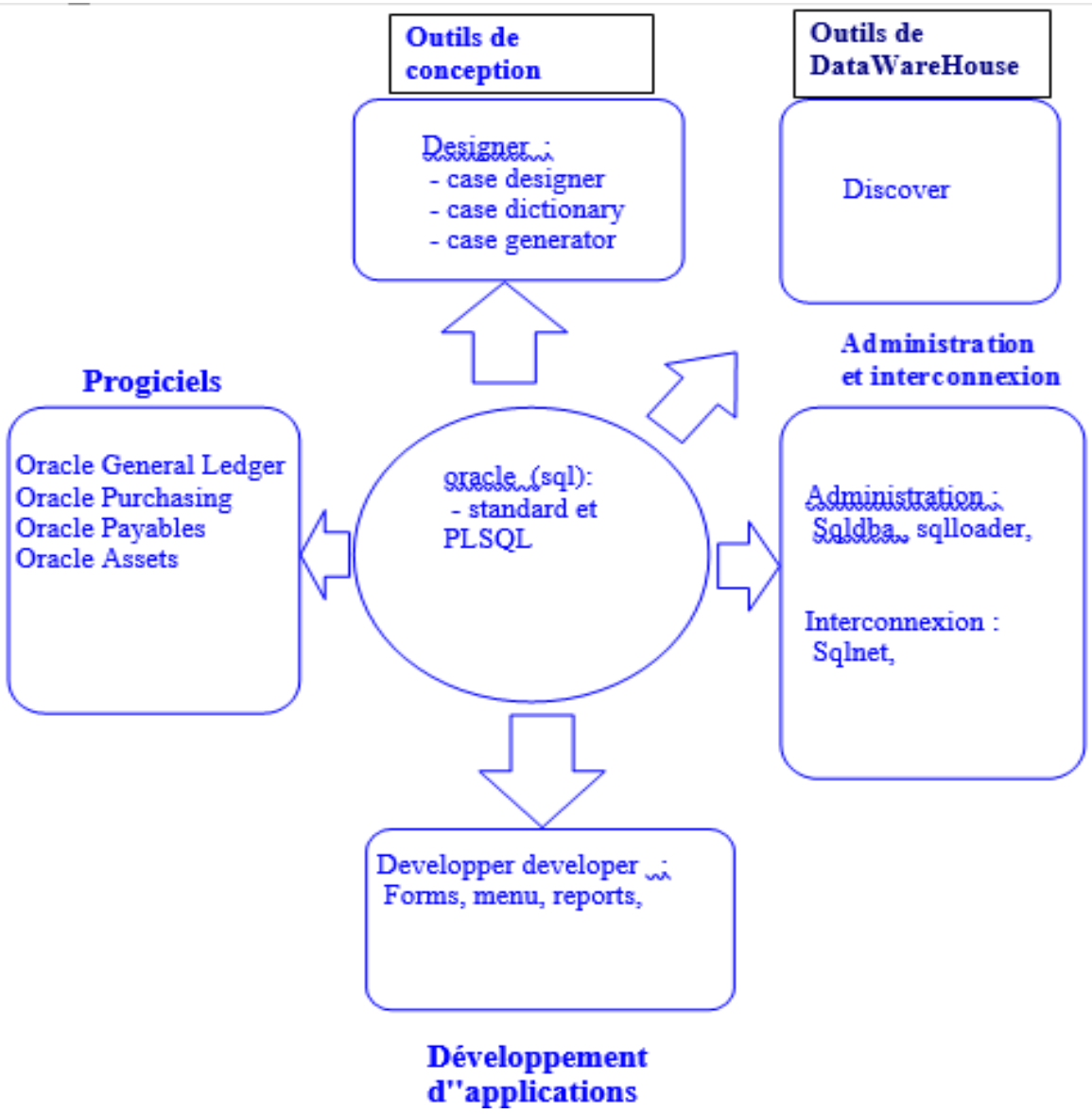
  pour :

    * 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.

1.3. 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>]
[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 (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
         .
16 ligne(s) sélectionnée(s).
```

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;
```

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;
```

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
```

2.2. Indépendance logique externe (suite ...)

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 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		nom de colonne
constante	OPERATEUR	constante
expression		expression

2.4. Opération de sélection

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

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

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 '*';
```

2.4.3. 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 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 Colonne
Constante Opérateur Constante
Fonction 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;
```

2.5.1. Les expressions

Colonne Colonne
Constante Opérateur Constante
Fonction 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;
```

2.5.1. Les expressions (suite ...)

Exemple 3 :

ajouter 3 jours à une date

'08-DEC-20' + 3 = '11-DEC-20'

Exemple 4 :

enlever 3 jours à une date

'11-DEC-20' - 3 = '08-DEC-20'

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).
```

5.2. Les fonctions

Fonctions numériques

- ABS(n) : valeur absolue de n
- MOD(m,n) : reste de la division de m par 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('TETE','E', 'O') = 'TOTO'

REPLACE('TATA','T') = 'AA'

2.5.2. Les fonctions (suite ...)

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

Exemples :

REPLACE('TETE','E', 'O') = 'TOTO'

REPLACE('TATA','T') = 'AA'

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-20'

ROUND(sysdate,'MM') = '01-JUL-20'

ROUND(sysdate + 4 ,'MM') = '01-AUG-20'

ROUND(sysdate,'YY') = '01-JAN-21' avec sysdate = 26/04/2021

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

Exemples :

TRUNC(sysdate,'MM') = '01-JUL-20'

TRUNC(sysdate + 4 ,'MM') = '01-JUL-20'

TRUNC(sysdate,'YY') = '01-JAN-21' avec sysdate = 26/04/2021

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;
```

==> 26 04 2021 15 :30

FIN COURS J 1

DATE : 26/04/2021

Horaire : 15H - 18H

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 MARS 2021 A 15 : 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 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
-------- ------------------------------
26/04/21 SCOTT
```

EXERCICES 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.

8. 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.

<u>Décomposition de la jointure :</u>

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 ambigu
- 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.

2.9. 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).

2.12. 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 correspondan

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 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. Le partitionnement

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 HAVIN joue le rôle de filtre pour la clause GROUP BY. L'exécution de la clause HAVING sera effect 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 7

"Pour chaque ville de localisation d'avions de la compagnie (sauf "Paris") donner le nombr
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 qu
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.

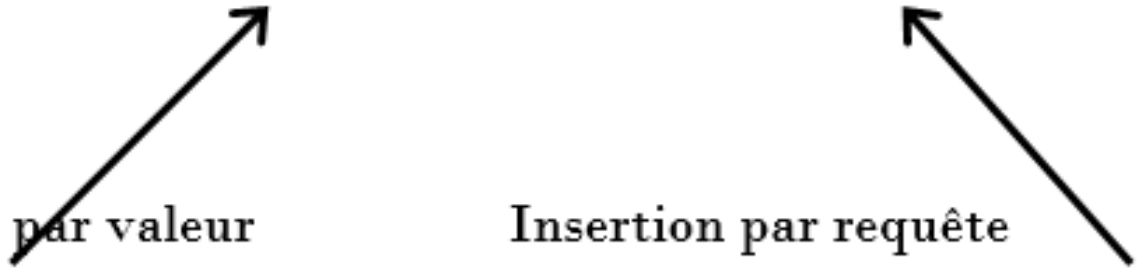
Commandes :

- 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 ;



3.1. Insertion de lignes

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%

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

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

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 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 précédents 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

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. Création d'une table

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

[contrainte_table]
)]
| as subquery};

FIN COURS J 2

DATE : 27/04/2021

Horaire : 15H - 18H

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)

4.1.1. 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	

4.1.1. 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
    );
```

4.1.1. 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#)
    );
```

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

Remarques :

- les mots clés PRIMARY KEY et REFERENCES permettent définir des contraintes d'intégrité d'entité et de référence.

- 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).

4.1.2. Création d'un index

Les index permettent 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, ...) ;

EXERCICES 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 (avec la clause PRIMARY KEY) sur le numéro du pilote et la date de formation.

Créer un index unique sur la colonne PLNOM de PILOTE. Que constatez vous.

Créer également un index sur la colonne AVTYPE 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

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

SUPPRESSION

- on ne peut supprimer une colonne

- on peut supprimer une contrainte par son nom

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 ...)

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 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 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_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_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

- 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$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 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 snapshot : 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.

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"

EXERCICES 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 ;

EXERCICES 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. Notion de 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.

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)*

Intégrité de domaine

Note : - avec oracle 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 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 peut être simulée grâce aux vues avec la clause WITH CHECK OPTION qui assure la vérification de la contrainte.

Note : sous Oracle , 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 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)*

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

"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 ?

7.GESTION UTILISATEURS, DROITS D'ACCÈS ET OBJETS DE SCHÉMA
(contrôle d'acces des utilisateurs)

7.1. Les privilèges système

7.1.1. Qu'est ce qu'un privilège

Les privilèges sont des droits pour exécuter des requêtes
Le plus haut niveau de privilèges sont des privilèges DBA, il a la possibilité de donner aux utilisateurs l'accès à la base de données.
Les utilisateurs doivent posséder des privilèges système pour se connecter à la base de données, et les privilèges objets pour manipuler des données.

7.1.2. Les privilèges DBA

Il existe plus de mille privilèges pour les utilisateurs et les rôles Privilège système
Opération autorisée CREATE USER Autorise de créer des utilisateurs
DROP USER Autorise de supprimer des utilisateurs
Autorise de supprimer toutes les tables dans tous les schémas
DROP ANY TABLE
BACKUP ANY TABLE
Autorise de sauvegarder toutes les tables dans tous les schémas.
SELECT ANY TABLE
Autorise d'effectuer les requêtes SELECT dans tous les schémas.
CREATE ANY TABLE
Autorise de créer des tables dans tous les schémas.

7.1.3. Créer un utilisateur

Un DBA peut créer des utilisateurs en utilisant la requête
CREATE USER.
Lorsqu'un utilisateur est créé, il ne possède aucun privilège. Le DBA doit lui donner des privilèges souhaités.
CREATE USER utilisateur
IDENTIFIED BY motdepasse;

7.1.4. Les privilèges système accordés à un utilisateur

Lorsque le DBA a créé un utilisateur, il lui donne des privilèges.
Exemple : Le DBA donne à l'utilisateur la possibilité de se connecter à la base de données. Ce privilège est donné grâce à CREATE SESSION.
GRANT privilege [,privilege, ...]
TO
user [, user, role, PUBLIC ...];

Privilège Opération autorisée

CREATE SESSION Autorise à se connecter sur la base de données
CREATE TABLE Autorise à créer des tables
CREATE SEQUENCE Autorise à créer des séquences
CREATE VIEW Autorise à créer des vues
Autorise à créer des procédures, des fonctions ou CREATE PROCEDURE des packages

7.1.5. Accorder un privilège

Pour accorder un privilège il faut suivre les étapes suivantes :

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

7.1.5. Accorder un privilège

Exemple :
SQL> CREATE USER scott2
2 IDENTIFIED BY tiger2;
Utilisateur crée.

SQL> GRANT CREATE SESSION
2 TO scott2;
Autorisation de privilèges (GRANT) acceptée.

SQL> GRANT CREATE TABLE
2 TO SCOTT2;
Autorisation de privilèges (GRANT) acceptée.

7.1.6 Créer et accorder un privilège à un rôle

Créer et accorder un privilège à un rôle
Un rôle est un ensemble de privilèges CREATE ROLE nomrole;
Lorsque le rôle est créé le DBA vous devez utiliser la requête GRANT pour assigner ce rôle aux utilisateurs.

Exemple:
SQL> CREATE ROLE   manager;

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

Autorisation de privilèges (GRANT) acceptée.

7.1.6 Créer et accorder un privilège à un rôle

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

7.1.7. Modification de mot de passe

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

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

Exemple :

```
SQL> ALTER USER     scott2
2  IDENTIFIED BY   oracle;
Utilisateur modifié.
```

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

7.2 Les privilèges objet

7.2.1. Les privilèges objet

Un privilège objet donne le droit d'effectuer des opérations sur des tables, vues, séquences ou procédures spécifiques.
Voici le tableau des privilèges pour différents objets.

Privilège objet	Table	Vue	Séquence	Procédure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X			
INSERT	X	X		
REFERENCES	X	X		
SELECT	X	X	X	
UPDATE	X	X		

7.2 Les privilèges objet

7.2.2. Accorder les privilèges

Le propriétaire d'un objet peut donner n'importe quel privilège objet à un autre utilisateur grâce à la requête GRANT Si cette requête possède l'option WITH GRANT OPTION, le nouvel utilisateur pourra donner le privilège objet à un autre utilisateur

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

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

7.2 Les privilèges objet

7.2.2. Accorder les privilège

Exemple :

```
SQL> GRANT  UPDATE
2  ON      emp
3  TO      scott;
```

Autorisation de privilèges (GRANT) acceptée.

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

7.2 Les privilèges objet

7.2.2. Accorder les privilège

Utilisation des mots clés WITH GRANT OPTION et PUBLIC

Mot clé WITH GRANT OPTION

Le privilège accordé avec la clause WITH GRANT OPTION donne la possibilité au nouvel utilisateur d'accorder les privilèges sur cet objet aux autres utilisateurs Si vous enlevez ensuite le privilège à cet utilisateur, tous les utilisateurs à qui il aura donné ce privilège se le verront enlevé aussi de manière automatique.

Exemple :

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

Autorisation de privilèges (GRANT) acceptée.

Explication

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

7.2 Les privilèges objet

7.2.2. Accorder les privilège

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

Exemple :

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

Autorisation de privilèges (GRANT) acceptée

7.2 Les privilèges objet

7.2.2. Accorder les privilège

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

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

7.2 Les privilèges objet

7.2.3. Retirer des privilège

Vous pouvez retirer des privilèges en utilisant la requête REVOKE

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

CASCADE CONSTRAINTS: Supprime toutes les contraintes d'intégrité référencée

Exemple :

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

Suppression de privilèges (REVOKE) acceptée.
```

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

8. L'environnement SQL*PLUS

8.1.Les variables de substitution

Lorsqu'une requête est exécutée un certain nombre de fois avec des valeurs différentes à chaque fois,la requête doit être modifiée et lancée autant de fois qu'il y a de valeurs différentes.

Les variables de substitutions serviront à saisir les valeurs de l'utilisateur à chaque lancement de la requête au lieu de modifier manuellement les valeurs.
Les variables de substitution sont des contenants dans lesquels sont stockées temporairement des valeurs.

L'utilisation des variables de substitution en SQL*Plus consiste à demander à l'utilisateur d'entrer une valeur qui sera substituée à la variable correspondante.
Une variable de substitution est une variable définie et nommée par le programmeur (celui qui écrit la requête).
Le nom de variable est précédé d'un à deux '&'.

8 L'ENVIRONNEMENT SQL*PLUS

8.1. variable de substitution

Une variable de substitution peut être placée n'importe où dans un ordre SQL exceptée en tant que
premier mot de l'ordre. Une variable de substitution ne peut pas remplacer une clause SELECT.

7.1.1. Utilisation d'esperluette &

Si la variable n'a pas de valeur ou si elle n'existe pas encore, SQL*Plus demandera à l'utilisateur de saisir une valeur pour cette variable à chaque fois qu'il l'a rencontrera dans un ordre SQL.

& user_variable pour les valeurs de type numérique

'& user_variable' pour les valeurs de type date et chaîne de caractères

8 L'ENVIRONNEMENT SQL*PLUS

8.1. variable de substitution

Exemple :

```
SQL>        SELECT        dname, deptno
  2         FROM          dept
  3         WHERE         deptno=&departement;

Entrez une valeur pour departement : 10

ancien    3 : WHERE deptno=&departement
nouveau   3 : WHERE deptno=10

DNAME              DEPTNO
-------------- ----------
ACCOUNTING             10
```

Explication : Cette requête affiche le nom et le numéro du département correspondant au numéro de département saisi par l'utilisateur (10).

7.1.2 Substitution de chaînes de caractères et de dates.

Rappel : Dans une clause WHERE
, les valeurs de type date et chaîne de caractères doivent être entre simples côtes.
Cette règle s'applique également aux variables de substitution. Si la valeur qu'elle substitue est du type date ou chaînes de caractères, la variable doit être placée entre simples côtes.
Par contre, lorsque l'utilisateur saisit la valeur, il ne doit pas mettre de simples côtes

Exemple :

```
SQL>        SELECT        ename, deptno, sal*12
  2         FROM          emp
  3         WHERE         job = '&job_title';

Entrez une valeur pour job title : ANALYST

ancien    3 : WHERE job = '&job_title'
nouveau   3 : WHERE job = 'ANALYST'

ENAME           DEPTNO     SAL*12
---------- ---------- ----------
SCOTT               20      36000
FORD                20      36000
```

L'environnement SQL*PLUS

Explication : Cette requête affiche le nom, le département et le salaire annuel des employés dont l'emploi est défini par l'utilisateur comme étant ANALYST. Dans le code SQL, la variable est entre simples cotes puisqu'elle substitue une chaîne de caractères. Ainsi, l'utilisateur n'a pas besoin de saisir les côtes.

On peut utiliser les fonctions
UPPER et LOWER sur des variables de substitution.
UPPER ('& user_variables ')
LOWER ('& user_variables ')
L'utilisation de ces fonctions permet de ne pas tenir compte de la casse de la valeur saisie par l'utilisateur.
Si la variable de substitution attend la saisie d'une date, elle doit être saisie au format par défaut DD-MON-YY

8. L'environnement SQL*PLUS

8.1.3 Utilisation de double ampersand &&

Si une variable est précédée de deux caractères '& ', alors SQL*Plus ne demandera la saisie de la valeur qu'une seule fois lors de l'exécution d'une requête.
&& user-variable pour les valeurs de type numérique '&& user-variable' pour les valeurs de type date et chaînes de caractères
On utilise le double '&', lorsqu'une variable est utilisé plusieurs fois dans une requête.

Exemple :

```
SQL>        SELECT        empno, ename, job, &&column name
  2         FROM          emp
  3         ORDER BY      &&column_name;

Entrez une valeur pour column_name : deptno

ancien    1 : SELECT empno, ename, job, &&column name
nouveau   1 : SELECT empno, ename, job, deptno

ancien    3 : ORDER BY &&column_name
nouveau   3 : ORDER BY deptno

     EMPNO ENAME      JOB              DEPTNO
---------- ---------- --------- ----------
      7782 CLARK      MANAGER            10
      7839 KING       PRESIDENT          10
      7934 MILLER     CLERK              10
      7369 SMITH      CLERK              20
      7876 ADAMS      CLERK              20
      7902 FORD       ANALYST            20
…
14 ligne(s) sélectionnée(s).
```

8 L'ENVIRONNEMENT SQL*PLUS

8.1. variable de substitution

Explication : L'utilisateur est appelé qu'une seule fois à saisir la variable column_name qui apparaît
deux fois dans l'ordre SQL. La valeur saisie par l'utilisateur (deptno) est utilisée pour les deux apparitions de la variable dans le code.
La valeur est stockée dans la variable jusqu'à la fin de la session ou jusqu'à ce qu'elle soit indéfinie

8.2 Définition des variables de substitution

La commande **DEFINE** est utilisée pour créer et définir des variables utilisateur.

Fonction	Définition
DEFINE *variable* = *value*	Créer la variable utilisateur *variable* de type CHAR et lui assigne la valeur *value*.
DEFINE *variable*	Affiche la variable *variable*, sa valeur et son type de données.
DEFINE	Affiche toutes les variables utilisateurs, leur valeur et leur type de données.

Exemple :

```
SQL>        DEFINE deptname = sales
SQL>        DEFINE deptname
DEFINE DEPTNAME         = "sales" (CHAR)
```

8 L'ENVIRONNEMENT SQL*PLUS

8.2 Définition des variables de substitution

Explication : L'utilisateur est appelé qu'une seule fois à saisir la variable column_name qui apparaît deux fois dans l'ordre SQL. La valeur saisie par l'utilisateur (deptno) est utilisée pour les deux apparitions de la variable dans le code.
La valeur est stockée dans la variable jusqu'à la fin de la session ou jusqu'à ce qu'elle soit indéfinie.

8.2 Définition des variables de substitution

La commande **DEFINE** est utilisée pour créer et définir des variables utilisateur.

Fonction	Définition
DEFINE *variable* = *value*	Créer la variable utilisateur *variable* de type CHAR et lui assigne la valeur *value*.
DEFINE *variable*	Affiche la variable *variable*, sa valeur et son type de données.
DEFINE	Affiche toutes les variables utilisateurs, leur valeur et leur type de données.

Exemple :

```
SQL>        DEFINE deptname = sales
SQL>        DEFINE deptname
DEFINE DEPTNAME         = "sales" (CHAR)
```

Explication : La première commande

DEFINE définie la variable deptname et lui attribue la valeur « sales » . La deuxième commande DEFINE

affiche la variable deptname , sa valeur (sales) et son type de données (CHAR).

Exemple :

```
SQL>        SELECT        *
  2         FROM          dept
  3         WHERE         dname = UPPER('&deptname');

ancien    3 : WHERE dname = UPPER('&deptname')
nouveau   3 : WHERE dname = UPPER('sales')

    DEPTNO DNAME          LOC
---------- -------------- -------------
        30 SALES          CHICAGO
```

Explication : La première commande DEFINE définie la variable deptname et lui attribue la valeur « sales » . La deuxième commande DEFINE affiche la variable deptname , sa valeur (sales) et son type de données (CHAR).

Exemple :

```
SQL>        SELECT        *
  2         FROM          dept
  3         WHERE         dname = UPPER('&deptname');

ancien    3 : WHERE dname = UPPER('&deptname')
nouveau   3 : WHERE dname = UPPER('sales')

    DEPTNO DNAME          LOC
---------- -------------- -------------
        30 SALES          CHICAGO
```

ORACLE DATABASE :

ORACLE SQL et l'ENVIRONNEMENT SQL*PLUS

Merci pour votre attention

Questions