CHAPITRE 2

LE LANGAGE SQL

Généralités :

I) Les types de données :

1) les chaînes de caractères :

. char(i)

. varchar(i)

leur longueur ne doit pas excéder 255 caractères.

. long

permet de stocker des chaînes de caractères d'une longueur maxi de 64 ko. Une table ne peut comporter qu'une seule colonne de type *long*. Une telle colonne ne peut être utilisée dans une expression, une clause *where*, une indexation.

2) les valeurs numériques:

de 10^{-129} à $9.99\ 10^{124}$

. number[(précision,[échelle])]

avec précision de 1 à 38

échelle de -84 à 127

3) le type date :

. date

permet de stocker une date et une heure

4) les séquences de caractères graphiques :

. raw

. longraw

ces types sont utilisés pour stocker les séquences de caractères graphiques. Sous SQL*PLUS une telle donnée sera affichée sous forme hexadécimale. L'insertion se fera également en donnant une valeur hexadécimale.

Ex: *insert into escape (seq) values ('1B5A');*

5) le type rowid :

. rowid

pseudo colonne correspondant à l'adresse d'une ligne dans une table.

Structure du rowid : bloc.ligne.fichier_de_la_base

II Les valeurs nulles :

NULL est l'état qui représente une valeur inconnue ou inapplicable dans une colonne.

Les colonnes, par défaut, ont la possibilité d'être mises à NULL.

Une valeur nulle dans un champ numérique n'est pas équivalent à 0.

La valeur de NULL+500 est NULL.

La valeur de GREATEST(NULL,500) est NULL.

Une chaîne de caractères de longueur 0 est interprétée comme une valeur nulle.

III) Les opérateurs :

1) les opérateurs mathématiques :

Utilisés avec les types numériques, ils génèrent un résultat numérique:

. + , -	plus , moins (unaire)
. **	exponentiation
. * , /	multiplication , division
. + , -	addition , soustraction

Remarque: ces 2 derniers opérateurs (+,-) peuvent être utilisés avec le type date sur certains SGBD.

2) les opérateurs de comparaison :

Ils génèrent un résultat logique :

. =	égal à
. != ou <>	différent de
. >	supérieur à
. >=	supérieur ou égal à
. <	inférieur à
. <=	inférieur ou égal à

3) les opérateurs logiques :

Ils génèrent un résultat logique :

. not	non (unaire)
. or	ou (binaire)
. and	et (binaire)

4) les prédicats like, between, in, any, all, exists, is NULL:

Ils génèrent un résultat logique :

. LIKE permet de comparer 2 chaînes de caractères dont l'une utilise un ou plusieurs caractères de remplacement.

. BETWEEN AND retourne une valeur vraie si l'expression appartient à un intervalle.

. IN retourne une valeur vraie si l'expression est égale à un des éléments de la liste ou un des tuples résultant de la sous-requête exprimée.

. ANY ou ALL doivent être précédés par un opérateur de comparaison et retournent une valeur vraie en fonction de l'opérateur de comparaison utilisé et des tuples résultant de la sous-requête exprimée.

. EXISTS retourne une valeur vraie si l'ensemble des tuples résultant de la sous-requête exprimée n'est pas vide.

.IS NULL retourne une valeur vraie si l'expression a pour valeur NULL.

IV) Les caractères de remplacement :

. % remplace un ensemble de caractères

ex : nom LIKE 'K%'

. _ remplace un et un seul caractère

V) Règles sur le nom des objets d'une base de données :

Les noms des *tables, vues, noms de correlation, colonnes, index, variables* doivent:

- ne pas excéder 30 caractères de longueur
- commencer par une lettre
- être composé de caractères ∈ {a-z,0-9,_,$,#}
- ne pas faire partie de l'ensemble des mots réservés
- ne pas être identique à un objet de même type (cad 2 objets de même type ne peuvent pas porter le même nom)

Présentation de quelques commandes SQL :

Conventions :

Dans la syntaxe formelle d'une commande :

- argument en majuscule : terme du langage

- argument en minuscule : argument à définir par l'utilisateur

- argument entre [ ] : argument optionnel

- argument entre { } : argument pouvant être itéré zéro ou plusieurs fois.

- / : ou

Create table (1ère forme)

Syntaxe simplifiée :

CREATE TABLE nom_de_table
(nom_de_colonne format{,nom_de_colonne format})

où format suit une des syntaxes suivantes :
type_de_données [NOT NULL]

Description :

L'ordre create table permet la création d'une table de la base de données.

NOT NULL une erreur est spécifiée si aucune valeur n'est entrée.

Exemples :

```
create table entete
  (ncom       number(5),
   date       date,
   nfou       char(2),
   statut     char(1) );

create table entete
  (ncom       number(5) not null,
   date       date,
   nfou       char(2)  not null,
   statut     char(1) );
```

Select (1ère partie) ou:

INTERROGATION D'UNE TABLE UNIQUE

Syntaxe simplifiée :

```
SELECT [DISTINCT] expression [nom_de _colonne]
        {,expression [nom_de_colonne]}
    / *
    FROM table
    [WHERE condition]
    [ORDER BY colonne [ASC/DESC]
        {,colonne [ASC/DESC]}
```

Description :

L'ordre select est l'ordre qui permet d'interroger la base de données; il permet d'obtenir une relation à partir d'une ou de plusieurs autres relations.

Exemples :

```
select *
    from fournisseurs;

select nfou
    from fournisseurs
    where nomfou = 'Fnac Rennes';
```

```
select refpiece,nfou
        from prix
        where prix > 100;

select refpiece,nfou
        from prix
        where prix between 100 and 150;

select refpiece,nfou
        from prix
        where prix is NULL;

select nomfou
        from fournisseurs
        where nomfou like 'F%';

select nfou, refpiece reference, prix*1.206 PTTC
        from prix
        where nfou='27' or nfou ='06';

select nfou, refpiece reference, prix*1.206 PTTC
        from prix
        where nfou in ('27','06');

select distinct ville
        from fournisseurs;

/*  Tri des données */
    select distinct ville,nomfou
            from fournisseurs
            order by ville,nomfou desc;
```

Create table (2ème forme)

Syntaxe simplifiée :

```
CREATE TABLE nom_de_table
      [(nom_de_colonne {,nom_de_colonne })]
      AS sous_select
```

Description :

Cette forme de l'ordre create table permet la création d'une table à partir d'une table existante.

Exemple :

```
create table lyonnais
      as select nfou, nomfou, adr, c_post, ville
      from fournisseurs
      where region = 'RHA';
```

Create view

Syntaxe simplifiée :

```
CREATE VIEW nom_de_vue
      [(nom_de_colonne {,nom_de_colonne })]
      AS sous_select
```

Description :

Create view permet la création d'une vue appelée aussi table virtuelle.

Exemple :

```
create view fou_lyon
      as select nfou, nomfou, adr, c_post, ville
      from fournisseurs
      where region = 'RHA';
```

Drop

Syntaxe :

DROP TABLE nom_de_table {,nom_de_table}
DROP VIEW nom_de_vue {,nom_de_vue}
DROP INDEX nom_d'index {,nom_d'index}

Description :

Drop permet la destruction des objets spécifiés.

Exemple :

drop view fou_lyon;

Insert (1ère forme)

Syntaxe simplifiée :

```
INSERT INTO nom_de_table (colonne {,colonne})
    VALUES (expression {,expression})
```

Description :

Cette forme de la commande permet l'insertion d'une ligne dans la table spécifiée.

Exemple :

```
insert into prix
    values ('51180','06','10120Q',0.60);

insert into prix (prix, refpiece, nfou)
    values (NULL,'51173','30003RV');
```

Delete

Syntaxe simplifiée :

```
DELETE FROM nom_de_table
    [WHERE condition]
```

Description :

Cette commande permet l'effacement des lignes de la table qui répondent à la condition spécifiée.

Exemples :

```
delete from prix
    where reffou='10120Q';

delete from prix
    where prix > 100;
```

Update

Syntaxe simplifiée :

```
UPDATE nom_de_table
    SET colonne=expression {,colonne=expression}
    [WHERE condition]
```

Description :

Cette commande remplace les valeurs des colonnes spécifiées par les expressions données pour toutes les lignes qui satisfont à la condition (mise à jour des données).

Exemples :

```
update prix
    set prix = 1305
    where reffou = '7500P';

update prix
    set prix = prix*1.1
    where nfou = '27';
```

LES TRANSACTIONS

Une transaction multi-instructions est un ensemble d'instructions dont tout, ou partie, peut être exécutée ou annulée.

Pendant une transaction :

- vous voyez la base comme si vous étiez le seul utilisateur; personne ne peut changer une valeur que vous avez lue ou écrite avant la fin de votre transaction.
- vos mises à jour sont invisibles pour les autres; personne ne peut lire vos mises à jour avant la fin de la transaction.
- des mises à jour incomplètes ne peuvent arriver; si une panne système survient les données originales sont restaurées.
- vous pouvez annuler la transaction, alors les données précédant le début de la transaction sont restaurées

Commit

Syntaxe :

COMMIT

Description :

Cette commande valide la transaction en cours, et une nouvelle est automatiquement lancée.

Exemple :

```
insert into prix (refpiece, nfou, reffou, prix)
      values ('51180','06','10120Q',0.60);
select * from prix;
commit;
```

Rollback

Syntaxe :

ROLLBACK [TO point-de-sauvegarde]

Description :

Cette commande annule tout ou partie de la transaction en cours; si un point de sauvegarde est spécifiée, la transaction est annulée jusqu'à ce point.

Exemple :

```
update prix
      set prix = prix*1.1
      where nfou = '27';
select * from prix;
rollback;
```

Savepoint

Syntaxe :

SAVEPOINT point-de-sauvegarde

Description :

Cette commande déclare un point de sauvegarde à l'intérieur d'une transaction; ce point de sauvegarde permettra alors une annulation partielle de la transaction. Tous les points de sauvegarde sont rendus inactifs quand la transaction est terminée.

Exemple :

```
insert into prix (refpiece, nfou, reffou, prix)
      values ('51180','06','10120Q',0.60);
savepoint paris;
insert into prix (refpiece, nfou, reffou, prix)
      values ('11230','270','12450P',570);
savepoint rennes;
insert into prix (refpiece, nfou, reffou, prix)
      values ('51180','06','10120Q',0.60);
rollback to rennes;
select * from prix;
rollback to paris;
commit;
```

Select (2ème partie) ou

OPERATION DE JOINTURE

Syntaxe simplifiée :

```
SELECT [DISTINCT] expression [nom_de _colonne]
                    {,expression [nom_de_colonne]}
        / *
        FROM table [nom_de_corrélation]
                    {,table [nom_de_corrélation]}
        [WHERE condition]
        [ORDER BY colonne [ASC/DESC]
                            {,colonne [ASC/DESC]}]
```

Description :

Un nom de corrélation permet à la table d'être utilisée sous un autre nom; ce nom de corrélation n'est valide que dans la requête où il a été défini.

Exemples :

Visualiser les numéros et les noms des fournisseurs:

```
select fournisseurs.nfou, fournisseurs.nomfou
    from fournisseurs;
```

est équivalent à :

```
select f.nfou, f.nomfou
    from fournisseurs f;
```

Visualiser les en-têtes de commandes accompagnés des noms des fournisseurs :

```
select nomfou, e.nfou, ncom, date, statut
    from fournisseurs f, entete e
    where f.nfou=e.nfou;
```

Donner le statut et le détail des commandes dont les numéros sont supérieurs à 5:

```
select statut, refpiece, qte, d.ncom
    from entete e, detail d
    where e.ncom=d.ncom and e.ncom>=5;
```

Trouver le prix des chaises et des bureaux;

```
select descr,prix
    from pieces, prix
    where pieces.refpiece=prix.refpiece
    and (descr like 'Chaise%'
        or descr like 'Bureau%');
```

est équivalent à :

```
select descr,prix
    from pieces, prix
    where pieces.refpiece=prix.refpiece and
    descr in ('Chaise de Bureau',
            'Bureau en Chene');
```

Donner le détail des commandes en attente (dont le statut est 'p'):

```
select d.*
    from entete e, detail d
    where e.ncom=d.ncom and statut='p';
```

est équivalent à :

```
select *
    from detail
    where ncom in
        (select ncom
            from entete
            where statut='p');
```

Insert (2ème forme)

Syntaxe simplifiée :

INSERT INTO nom_de_table [(colonne {,colonne})]
sous_select

Description :

Cette forme de la commande permet l'insertion d'une ou de plusieurs lignes dans la table spécifiée à l'aide du select exprimé. En d'autres termes, le résultat du select est inséré dans la table dont le nom suit le INTO.

Exemple :

Créer une table prix_pour_qte où les prix des articles pour les achats en quantité sont proposés avec respectivement 10%, 15% et 20% de réduction aux fournisseurs 06, 27 et 72.

```
create table prix_pour_qte
      as select refpiece, nfou, reffou, prix*0.9
            from prix
            where nfou='06';
insert into prix_pour_qte
      select refpiece, nfou, reffou, prix*0.85
            from prix
            where nfou='27';
insert into prix_pour_qte
      select refpiece, nfou, reffou, prix*0.8
            from prix
            where nfou='72';
```

Select (3ème partie) ou :

LES FONCTIONS D'ENSEMBLES ET LES GROUPES

Syntaxe simplifiée :

```
SELECT [DISTINCT] expression [nom_de _colonne]
                    {,expression [nom_de_colonne]}
        / *
        FROM table [nom_de_corrélation]
                    {,table [nom_de_corrélation]}
        [WHERE condition]
        [GROUP BY colonne {,colonne}
        [HAVING condition]]
        [ORDER BY colonne [ASC/DESC]
                        {,colonne [ASC/DESC]}]
```

Description :

Une fonction d'ensembles est utilisée dans une expression après un select, et permet de travailler sur une colonne et non plus sur une simple valeur.

Une fonction d'ensembles a pour syntaxe :

fonction_d'ensemble([DISTINCT/ALL] expression)

où DISTINCT indique que les valeurs dupliquées ne sont pas prises en compte et où ALL est la valeur par défaut.

Les fonctions d'ensembles sont :

Nom	Description
count	compte les occurences
sum	somme les valeurs numériques
avg	effectue la moyenne
max	renvoie la valeur maximale
min	renvoie la valeur minimale

Exemples :

Trouver le prix moyen des articles.

```
select avg(prix) as prix_moyen from prix;
```

Donner le nombre de différentes régions dont sont issus les fournisseurs.

```
select count(distinct region) from fournisseurs;
```

La clause GROUP BY permet d'établir des sous-ensembles à partir de l'ensemble de données initial.

Exemple :

Donner le prix le plus bas par pièce .

```
select refpiece, min(prix) from prix
    group by refpiece;
```

La clause HAVING permet de poser une condition sur les groupes établis.

Exemple :

Donner les prix moyens par pièce, pour les fournisseurs 27 et 06 et les pièces dont le prix moyen est supérieur à 1F.

```
select refpiece, avg(prix) from prix
    where nfou in ('27','06')
    group by refpiece
    having avg(prix)>1;
```

Gestion de nomenclatures

I) But et terminologie

Dans une table, il peut arriver qu'il existe un rapport de type groupe - élément entre deux entités de même type. Un groupe peut être considéré comme un élément à part entière de même que tout élément peut constituer un groupe.

Exemple: on isole cette partie de la table ENSEMBLE.

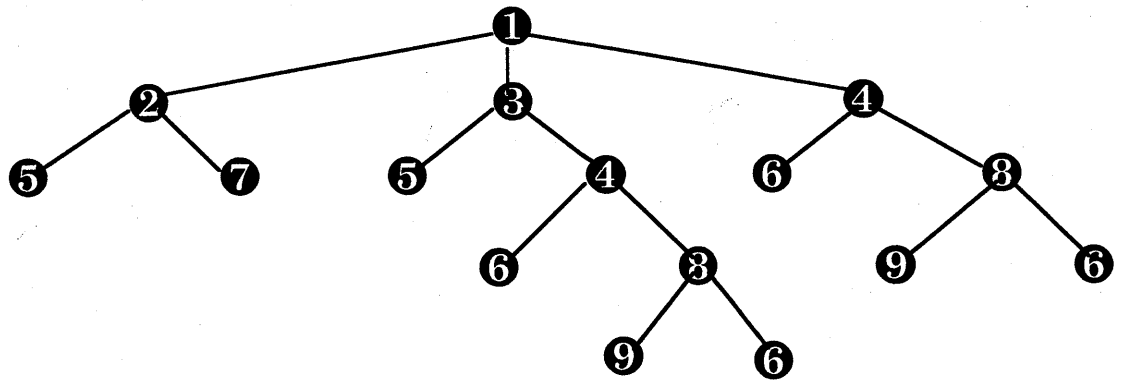
Elément	1	2	3	4	4	5	5	6	6	7	8	9
Groupe		1	1	1	3	2	3	4	8	2	4	8

En lisant à partir des éléments, on peut dire:
- 1 n'appartient à aucun groupe
- 2 appartient à 1
- 3 appartient à 1 ...

En lisant à partir des groupes et en effectuant une recherche, on peut dire:
- 1 est constitué de 2, 3, 4
- 2 est constitué de 5,7
- 3 est constitué de 4,5 ...

Ce rapport entre les deux entités peut se représenter sous la forme d'une arborescence.

EXPLOSION

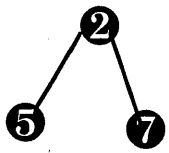


IMPLOSION

Cette représentation est beaucoup plus claire que la table elle-même et il est donc intéressant de pouvoir l'obtenir entièrement ou en partie.

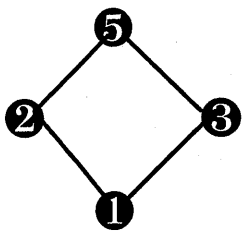
Par la suite, on cherchera à isoler une partie de cette arborescence en la parcourant soit:
- dans le sens descendant: **explosion**,
- dans le sens ascendant: **implosion**.

Exemples d'affichages recherchés:



a) explosion de l'élément 2

b) implosion de l'élément 7



c) implosion de l'élément 5

II) Fonction de gestion de nomenclature

Pour résoudre le problème posé, il faut donc une requête qui tient compte au moins:
- des données à sélectionner,
- du point de départ du parcours de l'arborescence,
- du sens de parcours de l'arborescence.

Eventuellement, on doit pouvoir également filtrer certaines données dans notre sélection.

La requête suivante résoud notre problème:

Le choix des éléments: **SELECT colonne(s)**
FROM table(s)
[WHERE critères de sélection]

Partie sélection des données dans l'explosion / implosion.

Le(s) point(s) de départ(s): **START WITH critère de départ**

Le sens de parcours: **CONNECT BY [PRIOR] composé = [PRIOR] composant**
[AND critères de constitution de l'arbre]

Partie constitution de l'explosion ou implosion

RQ: Le PRIOR est unique par expression et nécessaire.

Exemples (a, b, c se réfèrent aux exemples du I):

Requête	Traduction
a) SELECT element FROM Ensemble START WITH element=2 CONNECT BY PRIOR element=groupe	Choisir les elements dans Ensemble en commençant le parcours de l'element 2 de telle façon que le membre antérieur à element soit groupe
b) SELECT element FROM Ensemble START WITH element=7 CONNECT BY PRIOR groupe=element	Choisir les elements dans Ensemble en commençant le parcours de l'element 7 de telle façon que le membre antérieur à groupe soit element
c) SELECT element FROM Ensemble START WITH element=5 CONNECT BY PRIOR groupe=element	Choisir les elements dans Ensemble en commençant le parcours de l'element 5 de telle façon que le membre antérieur à groupe soit element

III) Améliorer l'affichage

Le mot clé LEVEL permet de connaître à tout moment le niveau où l'on se situe dans l'explosion ou l'implosion. On peut ainsi l'afficher pour se repérer ou mieux encore s'en servir pour une indentation en combinaison avec la fonction LPAD.

Exemple:

a)
```
SELECT LPAD(' ',LEVEL*2)||element FROM Ensemble
START WITH element=2
CONNECT BY PRIOR element=groupe
```

CHAPITRE 2

C - LES FONCTIONS SOUS SQL

Char Functions

Type	*Function*	*Value Returned*
N	ASCII(*char*)	ASCII value of first character of *char*.
C	CHR(*n*)	The character with ASCII value *n*.
C	INITCAP(*char*)	*char*, with first letter of each word capitalized.
N	INSTR(*char1*,*char2*[,*n*[,*m*]])	The position of the *m*th occurrence of *char2* in *char1*, beginning search at position *n*. If *m* is omitted, "1" is assumed. If *n* is omitted, "1" is assumed. Position is given relative to the first character of *char1*, even when *n* > 1.
N	LENGTH(*char*)	The length, in characters, of *char*.
C	LOWER(*char*)	*char*, with all letters forced to lower case.
C	LPAD(*char1*,*n*[,*char2*])	*char1*, left-padded to length *n* with the sequence of characters in *char2*, replicated as many times as necessary; if *char2* is omitted, with blanks.
C	LTRIM(*char*,*set*)	*char*, with initial characters removed up to the first character not in *set*.
C	RPAD(*char1*,*n*[,*char2*])	*char1*, right-padded to length *n* with the characters in *char2*, replicated as many times as necessary; if *char2* is omitted, with blanks.
C	RTRIM(*char*,*set*)	*char*, with final characters removed after the last character not in *set*.
C	SOUNDEX(*char*)	A char value representing the *sound* of the word(s) in *char*.
C	SUBSTR(*char*,*m*[,*n*])	A substring of *char*, beginning at character *m*, *n* characters long (if *n* is omitted, to the end of *char*).
C	TRANSLATE(*char*,*from*,*to*)	*char*, translated from the character set *from* to the character set to. Each character in *char* that appears in *from* is translated to the corresponding character from *to*.
C	UPPER(*char*)	*char*, with all letters forced to upper case.
C	USERENV(*char*)	Returns information about the user that is useful for writing an application-specific audit trail table. If *char* is 'ENTRYID', returns an available auditing entry identifier; if 'SESSIONID', returns user's auditing session identifier; and if 'TERMINAL', returns user's terminal's operating system identifier. LANGUAGE returns the spoken lenguage in use (such as 'ENGLISH').

Date Functions

Type	*Function*	*Value Returned*
D	ADD_MONTHS(*d*,*n*)	Date *d* plus *n* months.
D	LAST_DAY(*d*)	Date of last day of month containing *d*.
N	MONTHS_BETWEEN(*d*,*e*)	Number of months between dates *d* and *e*. If *d* is later than *e*, result is positive; if earlier, negative.
D	NEW_TIME(*d*,*a*,*b*)	Date and time in time zone *b* when date and time in time zone *a* are *d*. *a* and *b* are char expressions with the following meaning: AST, ADT — Atlantic Standard Time, Daylight Time. BST, BDT — Bering Standard Time, Daylight Time. CST, CDT — Central Standard Time, Daylight Time. EST, EDT — Eastern Standard Time, Daylight Time. GMT — Greenwich Mean Time. HST, HDT — Alaska-Hawaii Standard Time, Daylight Time. MST, MDT — Mountain Standard Time, Daylight Time. NST — Newfoundland Standard Time. PST, PDT — Pacific Standard Time, Daylight Time. YST, YDT — Yukon Standard Time, Daylight Time.
D	NEXT_DAY(*d*,*char*)	Date of first day of week named by *char* that is equal to or later than *d*.
N	TRUNC(*d*)	*d* with time of day truncated.

Conversion Functions

Type	*Function*	*Value Returned*
C	CHARTOROWID(*char*)	Converts a char value to a row ID.
W	HEXTORAW(*char*)	Converts a char value containing hexadecimal digits to a binary value (suitable for inclusion in a RAW column).
C	RAWTOHEX(*raw*)	Converts *raw* to a char value containing a hexadecimal number.
C	ROWIDTOCHAR(*rowid*)	Converts a row ID to a char value. The result of this conversion is 18 characters long.
C	TO_CHAR(*n*[,*fmt*])	
C	TO_CHAR(*d*[,*fmt*])	Converts *n* or *d* to a char value in the format specified by the char value *fmt*. For information about *fmt* see the section "Format Models," elsewhere in this chapter. If *fmt* is omitted, *n* is converted to a char value exactly long enough to hold the significant digits; *d* is converted to a char value in ORACLE's default date format, 'DD-MON-YY'.
D	TO_DATE(*char*[,*fmt*])	Converts a date from a char value to a date value. *fmt* is a char value specifying the format of *char*. For information about *fmt* see the section "Format Models," elsewhere in this chapter. If *fmt* is omitted, *char* must have the default date format, 'DD-MON-YY'.
D	TO_DATE(*n*,[*fmt*])	Converts a number into a date. *fmt* must be a format that when used in TO_CHAR, produces a number (e.g. 'J' or 'MM').
N	TO_NUMBER(*char*)	Converts *char*, a char value containing a number, to a number value.

Group Functions

Group functions are meaningful only in queries and subqueries.

DISTINCT makes a group function consider only distinct values of the expression; ALL makes it consider all values. For example, the DISTINCT average of 1, 1, 1, and 3 is 2; the ALL average is 1.5.

Type	*Function*	*Value Returned*
N	AVG([DISTINCT|ALL] *n*)	Average value of *n*, ignoring null values.
N	COUNT({[DISTINCT|ALL] *expr*|*})	Number of rows where the expression *expr* evaluates to something other than NULL. '*' makes COUNT count all selected rows.
N	MAX([DISTINCT|ALL] *expr*)	Maximum value of *expr*.
N	MIN([DISTINCT|ALL] *expr*)	Minimum value of *expr*.
N	STDDEV([DISTINCT|ALL] *n*)	Standard deviation of *n*, ignoring null values.
N	SUM([DISTINCT|ALL] *n*)	Sum of values of *n*.
N	VARIANCE([DISTINCT|ALL] *n*)	Variance of *n*, ignoring null values.

Other Functions

Type	*Function*	*Value Returned*
•	DECODE(*expr*, *search1*, *return1*, *search2*, *return2*, ... [*default*])	If *expr* equals any *search*, returns the following *return*; if not, returns *default*. If *default* is omitted and there is no match, NULL is returned. *expr* may be any data type; *search* must be the same type. The value returned is forced to the same data type as the first *return*.
•	DUMP(*expr*[, *radix*[, *start-position*[, *bytes*]]])	Displays the value of an expression in internal format.
•	GREATEST(*expr*,*expr*,...)	Returns the greatest of a list of values. All *exprs* after the first are converted to the type of the first before the comparison is done.
•	LEAST(*expr*,*expr*,...)	Returns the least of a list of values. All *exprs* after the first are converted to the type of the first before the comparison is done.
•	NVL(*x*,*expr*)	If *x* is null, returns *expr*; if *x* is not null, returns *x*. *x* and *expr* may be of any type. The type of the returned value is the same as the type of *x*.
N	VSIZE(*expr*)	Returns the number of bytes in ORACLE's internal representation of *expr*.

CHAPITRE 2

D - EXERCICES

EXERCICES (D1) : **Données de la base EXO**

1) Créer les tables EMPLOYES et DEPARTEMENT.

2) Entrer les données des tables EMPLOYES et DEPARTEMENT.

Soit la table '*département*' suivante:

deptno	dnom	dville
10	ventes	Lyon
20	recherche	Grenoble
30	formation	Lyon
40	développement	Paris
50	administration	Paris

Soit la table '*employés*' suivante:

empno	nom	deptno	fonction	chef	embauche	salaire	commission
0610	durand	30	formateur	2000	12/12/96	9500.00 F	
1010	leroy	50	directeur		21/01/99	20000.00 F	
2000	lumiere	30	coordonateur	1010	05/10/94	10800.00 F	
2050	souillard	10	resp. ventes	1010	12/06/95	14000.00 F	0.00 F
2051	caubert	10	vendeur	2050	21/01/98	8000.00 F	4000.00 F
2052	dangu	10	vendeur	2050	11/05/92	13000.00 F	5000.00 F
3021	dubois	50	secretaire	1010	21/01/99	7500.00 F	
5020	chevreuil	50	secretaire	0610	30/10/97	7900.00 F	
6010	legrand	30	vendeur	2000	06/07/97	7900.00 F	1020.00 F

EXERCICES (D2) : **Requêtes simples**

1) Liste des *vendeur* des départements 10 et 30.

2) Liste des employés qui ne sont pas *vendeur* et qui ont été embauchés en 1997 et 1998.

3) Liste des employés ayant un U et un A dans leur nom.

4) Liste des employés ayant deux E dans leur nom.

5) Liste des employés ayant une commission.

6) Liste des noms, numéros de département, fonctions et dates d'embauche triés par :
- numéro de département croissant
- ordre alphabétique des fonctions,
- ancienneté décroissante (les plus anciens d'abord).

7) Liste des différents emplois.

8) Que seraient les salaires du département 30 s'ils étaient augmentés de 7%.

9) Liste des employés dont la commission est supérieure au tiers du salaire.

EXERCICES (D3) : **Mise à jour et jointures**

1) Supprimer le département *recherche.*

2) Le département *développement* est transféré à *Lyon*. Mettez les données à jour.

3) Liste des employés travaillant dans le département *ventes*.

4) Liste des employés travaillant à *Lyon* et ayant un salaire supérieur à 10 000 F.

5) Noms et dates d'embauche des employés embauchés avant leur chef, avec le nom et la date d'embauche du chef.

6) Noms, villes et dates d'embauche des employés embauchés avant *Souillard.*

7) Liste des employés gagnant plus que 80% du salaire de leur chef.

8) Liste des employés du département *ventes* dont le salaire ajouté à la commission est supérieur à 75% du salaire de leur chef (ne pas oublier que la somme d'une valeur numérique et NULL donne un résultat NULL; utiliser en conséquence une fonction permettant d'obtenir le bon résultat).

EXERCICES (D4) : **Opérations ensemblistes et requêtes imbriquées**

1) Lister les numéros des employés n'ayant pas de subordonné.

2) Liste des employés embauchés le même jour que *Leroy*.

3) Liste des employés ayant le même chef que *Caubert*.

4) Liste des employés embauchés avant tous les employés du département 10.

5) Liste des employés ayant la même fonction et le même chef que *Dangu*.

6) Liste des employés du département *formation* embauchés le même jour que quelqu'un du département *ventes*.

7) Liste des employés ne travaillant pas dans le même département que leur chef.

EXERCICES (D5) : **Requêtes utilisant des fonctions**

1) Liste des noms des employés avec leur salaire tronqué au millier.

2) Liste des employés en remplaçant le nom par "****" dans le département 10.

3) Histogramme des salariés.

4) Noms des employés avec les dates de début du mois d'embauche.

5) Noms des employés et nombres de mois depuis l'embauche.

6) Salaires moyens en tenant compte des commissions.

7) Nombre d'employés dans chaque fonction.

8) Nombre d'employés dans chaque tranche de salaire (millier) .

9) Liste des employés ayant le salaire maximum de chaque département.

10) Fonction ayant le salaire moyen le plus bas.

EXERCICES (D6) : **Utilisation des vues**

1) Créer une vue des *secrétaires*. Tester.

2) Augmenter les salaires des *secrétaires* de 10% en utilisant cette vue.

3) Vérifier que les modifications ont été prises en compte dans la table *employes*.

4) Faire un ROLLBACK et regarder l'état de la table *employes*.

5) Créer une vue *rev_emp* (nom, revenu, département, ville); il est à noter que le *revenu* est constitué du *salaire* ajouté à la *commission*.

6) Faire la liste de tous les employés dont le revenu est supérieur à 10 000 F. Présenter les informations classées par département; dans chaque département les revenus apparaîtront de manière décroissante.

7) Créer une vue *stat_dept* (département, ville, nb_salariés, coût_revenus) basée sur la vue *rev_emp*.

8) Faire la liste des départements dont le nombre d'employés est supérieur à 2, en donnant pour chaque département son nom, sa ville et son nombre d'employés.

9) Supprimer la commission à *Caubert*.

10) Refaire les questions 6 et 8. Qu'observe-t-on?

EXERCICES (D7) : **Mise en forme de l'information**

1) Arbre hiérarchique de la société sous la forme :
Hiérarchie

.....

Leroy

Lumière

Durand

Chevreuil

Legrand

Souillard

Dangu

.....

2) Liste de la table des employés avec titre, total des salaires par fonction et total général.

3) Produire un état ayant la forme suivante :

- Société du Safran -
Salaires dans les départements par emploi

Fonction	Département 10	Département 30	Département 50	Total par emploi
Coordonateur	X	X	X	X
Formateur	X	X	X	X
Président	X	X	X	X
Resp. ventes	X	X	X	X
Secrétaire	X	X	X	X
Vendeur	X	X	X	X
	X	X	X	X

Rapport confidentiel

LES DONNEES DE LA BASE COMMANDES

FOURNISSEURS

nfou	nomfou	adr	ville	region	c_post
6	Gibert Jeune	25, bd St-Michel	Paris	IDF	75005
27	Fnac Rennes	12, Rue de Rennes	Paris	IDF	75006
72	Fournisseur Express	105, Avenue Rapp	Lyon	RHA	69000

ENTETE

NCOM	DATE	NFOU	STATUT
1	11-oct-97	27	c
2	12-oct-97	27	c
3	12-oct-97	6	c
4	07-nov-97	6	c
5	08-nov-97	72	p
6	25-nov-97	27	p

DETAIL

ncom	refpiece	qte
1	51173	5
1	51175	100
1	00001	12
2	51173	10
3	51173	5
4	00003	3
5	51173	100
5	57003	12
6	51175	100
6	51180	10

PIECES

refpiece	descr
00001	Chaise de Bureau
00003	Bureau en Chene
51173	Trombones n°2
51175	Bloc Papier Ligne Jaune
51180	Règle en Plastique
57003	Feutres Pointe Extra-Fine

PRIX

refpiece	nfou	reffou	prix
00001	27	29331X	175
00003	6	7500P	1299
51173	6	35000Q	0,57
51173	27	10179A	0,31
51173	72	43725Z	0,71
51175	27	73122Z	0,9
51180	27	73255Z	0,59