

Module n°2
Techniques de récupération des
données

ORACLE

Table des matières

1	AFFICHER DES DONNEES ISSUES DE PLUSIEURS TABLES	4
1.1	Accéder à plusieurs tables	4
1.1.1	Les types de jointures	4
1.1.2	Le produit cartésien	4
1.2	Récupérer des enregistrements avec une équi-jointure	5
1.2.1	Relier des tables avec une équi-jointure	5
1.2.2	Ajouter des conditions de recherche	6
1.2.3	Utiliser des alias de table	7
1.2.4	Relier plus de deux tables	8
1.3	Les autres types de jointures	9
1.3.1	Relier des tables avec une non équi-jointure	9
1.3.2	Relier des tables avec une jointure externe	10
1.3.3	Relier une table à elle-même avec une auto-jointure	11
2	LES FONCTIONS DE GROUPE	13
2.1	Les types de fonctions de groupe	13
2.1.1	Définition d'une fonction de groupe	13
2.1.2	Fonctions de groupe	13
2.1.3	Utilisations des fonctions de groupe	13
2.2	Créer des groupes de données	15
2.2.1	La clause GROUP BY	15
2.2.2	Groupeement sur plusieurs colonnes	16
2.2.3	Restreindre le résultat des groupes avec la clause HAVING	17
2.2.4	Requêtes invalides utilisant des fonctions de groupe	18
2.2.5	Les fonctions de groupe imbriquées	19
3	LES SOUS-REQUETES	20
3.1	Les sous-requêtes basiques	20
3.1.1	Les règles de conduite des sous-requêtes	20
3.1.2	Les types de sous-requête	20
3.2	Les sous-requêtes Single-Row	21
3.2.1	Ecriture d'une sous-requête single-row	21
3.2.2	Utilisation de fonctions de groupe dans des sous-requêtes single-row	22
3.2.3	Des sous-requêtes dans la clause HAVING	22
3.2.4	Les problèmes les plus courants lors de l'utilisation de sous-requêtes single-row	23
3.3	Les sous-requêtes Multiple-Row	24
3.3.1	Règles de conduite des sous-requêtes multiple-row	24
3.3.2	Ecriture de sous-requêtes multiple-row	24
3.3.3	Les valeurs nulles dans les sous-requêtes multiple-row	25
3.4	Les sous-requêtes Multiple-column	26
3.4.1	Ecritures de sous-requêtes multiple-column	26
3.4.2	Comparaisons de plusieurs colonnes	26
3.4.3	L'utilisation de sous-requêtes multiple-column dans la clause FROM	27
4	CREER DES RAPPORTS AVEC SQL*PLUS	29
4.1	Les variables de substitution	29
4.1.1	Caractéristiques des variables de substitution	29
4.1.2	Substitution de nombres	29
4.1.3	Substitution de chaînes de caractères et de dates	30
4.1.4	Utilisations des variables de substitution	31
4.1.5	Variable de substitution avec un double Ampersand	31
4.2	Les variables définies par l'utilisateur	32

4.2.1	La commande ACCEPT	32
4.2.2	Les Commandes DEFINE et UNDEFINE	33
4.3	Personnaliser l'environnement SQL*Plus	34
4.3.1	Les variables système et la commande SET	34
4.3.2	Les commandes de formatage SQL*Plus	35
4.3.3	La commande COLUMN	35
4.3.4	La commande BREAK	36
4.3.5	Les commandes TITLE	37
4.3.6	Exécuter un rapport formaté sous SQL*Plus	38

1 AFFICHER DES DONNEES ISSUES DE PLUSIEURS TABLES

1.1 Accéder à plusieurs tables

1.1.1 Les types de jointures

Pour afficher des données issues de plusieurs tables, il faut utiliser une condition appelée jointure. Une condition de jointure spécifie une relation existante entre les données d'une colonne dans une table avec les données d'une autre colonne dans une table. Cette relation est souvent établie entre des colonnes définies comme clé primaire et clé étrangère.

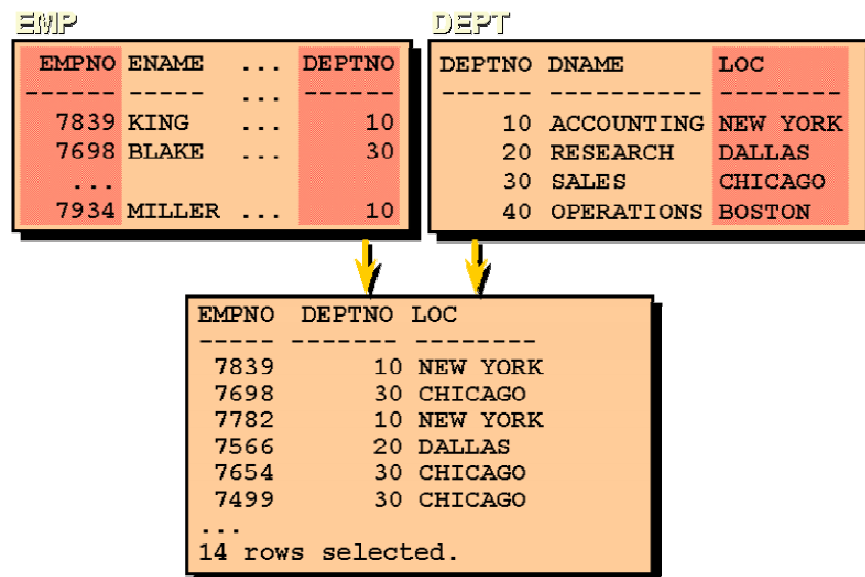


Figure 1 : Relation entre les valeurs contenues dans la colonne (clé étrangère) DEPTNO de la table EMP et la colonne (clé primaire) DEPTNO de la table DEPT

La condition de jointure doit être réalisée dans la clause **WHERE**.

```
SELECT      table1.column, table2.column
FROM        table1, table2
WHERE       table1.column1 = table2.column2 ;
```

Il existe quatre types de jointures :

- Equi-jointure (equijoin)
- Non equi-jointure (non-equijoin)
- Jointure externe (outer join)
- Auto jointure (self join)

1.1.2 Le produit cartésien

Un produit cartésien se produit lorsque :

- une condition de jointure est omise
- une condition de jointure est invalide
- tous les enregistrements de la première table sont liés à tous les enregistrements de la seconde table.

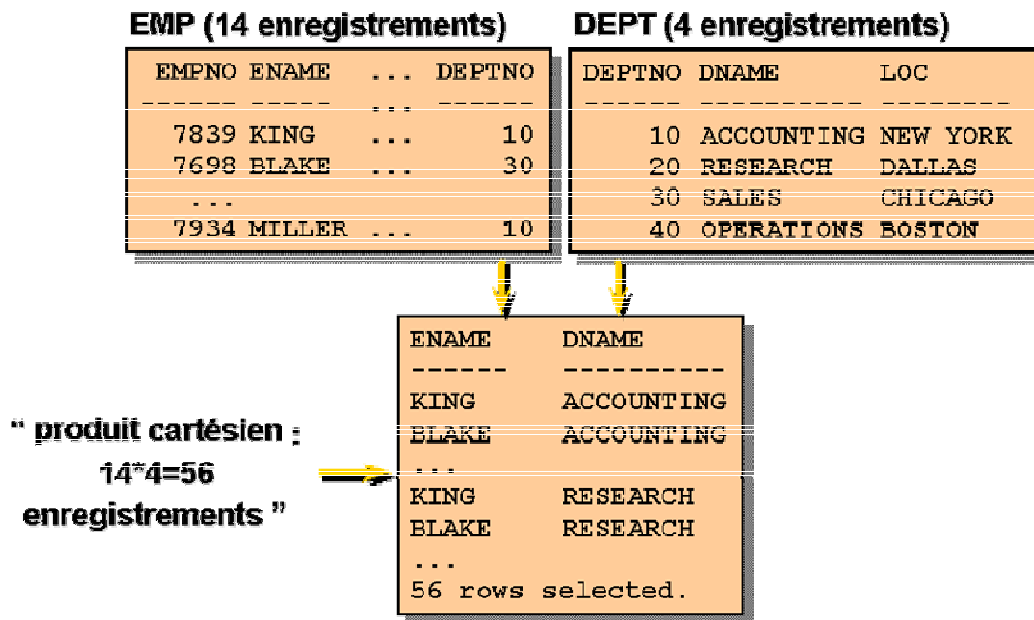


Figure 2 : Le produit cartésien des tables EMP et DEPT donnent 56 enregistrements

Le produit cartésien génère un grand nombre d'enregistrements dont le résultat est rarement très utile.

1.2 Récupérer des enregistrements avec une équi-jointure

1.2.1 Relier des tables avec une équi-jointure

Une équi-jointure est utilisée pour afficher des données provenant de plusieurs tables lorsqu'une valeur dans une colonne d'une table correspond directement à une valeur d'une autre colonne dans une autre table (cf. figure 3).

Les noms des colonnes doivent être qualifiés avec le nom de la table ou l'alias de la table à laquelle elles appartiennent afin d'éviter toute ambiguïté.

```
SELECT      table1.column, table2.column
FROM        table1, table2
WHERE       table1.column = table2.column;
```

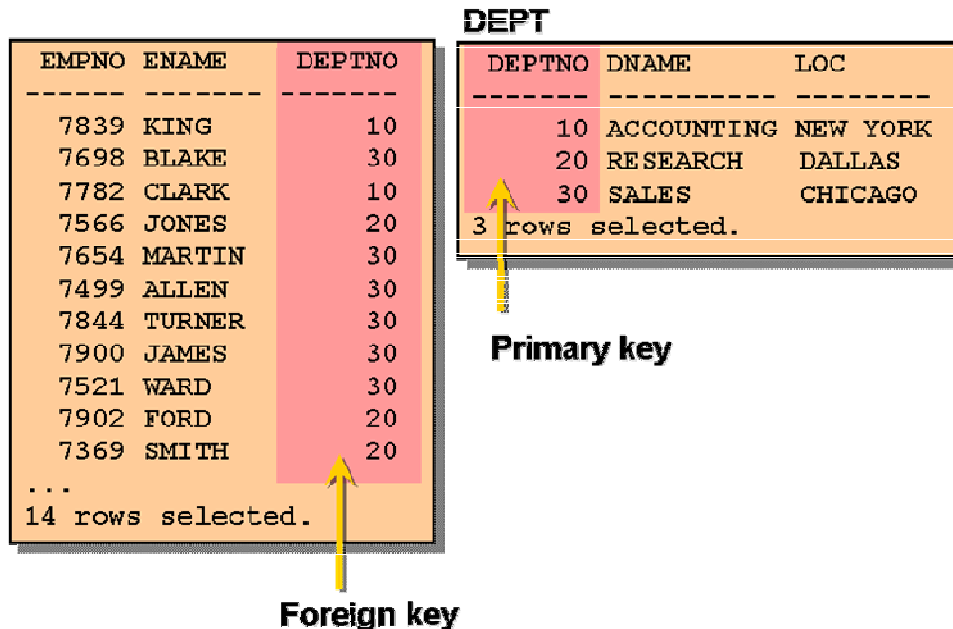


Figure 3 : La clé étrangère de la table EMP est liée à la clé primaire de la table DEPT

Exemple :

```
SQL> SELECT emp.empno, emp.ename, emp.deptno,
2         dept.deptno, dept.loc
3 FROM emp, dept
4 WHERE emp.deptno=dept.deptno;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS

...
14 rows selected.

Æ La requête affiche, pour chaque employé, son numéro (*emp.empno*), son nom (*emp.ename*), son numéro de département (*emp.deptno* et *dept.deptno*) et la localisation du département (*dept.loc*). La colonne DEPTNO de la table EMP est reliée à la colonne DEPTNO de la table DEPT. Pour chaque numéro de département de la colonne DEPTNO de la table EMP, la requête cherche la localisation correspondante dans la table DEPT.

1.2.2 Ajouter des conditions de recherche

Des conditions peuvent être ajoutées à la condition de jointure afin de restreindre les enregistrements.

```
SELECT          table1.column, table2.column
FROM            table1, table2
WHERE           table1.column = table2.column
[AND           (search_condition
OR             search_condition)];
```

Il est conseillé d'ajouter des parenthèses pour clarifier la requête.

EMP			DEPT		
EMPNO	ENAME	DEPTNO	DEPTNO	DNAME	LOC
7839	KING	10	10	ACCOUNTING	NEW YORK
7698	BLAKE	30	20	RESEARCH	DALLAS
7782	CLARK	10	30	SALES	CHICAGO
7566	JONES	20	3 rows selected.		
7654	MARTIN	30			
7499	ALLEN	30			
7844	TURNER	30			
7900	JAMES	30			
7521	WARD	30			
7902	FORD	20			
7369	SMITH	20			
...					
14 rows selected.					

Figure 4 : Restreindre les enregistrements à l'employé KING seulement à l'aide d'une condition supplémentaire dans la clause WHERE

Exemple :

```
SQL> SELECT emp.empno, emp.ename, emp.deptno,
2         dept.deptno, dept.loc
3 FROM emp, dept
4 WHERE emp.deptno=dept.deptno
5 AND emp.ename = 'KING' ;

EMPNO ENAME DEPTNO DEPTNO LOC
-----
7839 KING      10      10 NEW YORK

1 row selected.
```

1.2.3 Utiliser des alias de table

Pour alléger la requête, il est conseillé d'utiliser des alias pour les noms de table.

Exemple sans alias :

```
SQL> SELECT emp.empno, emp.ename, emp.deptno,
2         dept.deptno, dept.loc
3 FROM emp, dept
4 WHERE emp.deptno=dept.deptno;
```

Exemple avec des alias :

```
SQL> SELECT e.empno, e.ename, e.deptno,
2         d.deptno, d.loc
3 FROM emp e, dept d
4 WHERE e.deptno= d.deptno;
```

Remarques : les alias de table peuvent être utilisés pour améliorer les performances. En effet, ils raccourcissent le code SQL et utilisent donc moins de mémoire. (Le gain de performance est surtout visible sur les grosses requêtes).

1.2.4 Relier plus de deux tables

Pour joindre n tables ensemble, au moins $(n - 1)$ conditions de jointure sont nécessaires. Donc pour joindre trois tables, deux conditions de jointures doivent être écrites :

```
SELECT      table1.column, table2.column, table3.column
FROM        table1, table2, table3
WHERE       table1.column = table2.column
AND         table2.column = table3.column ;
```

Cette règle ne s'applique pas si une table possède une clé primaire concaténée (constituée de plusieurs colonnes). Dans ce cas, plus d'une colonne sont nécessaires pour identifier de manière unique chaque enregistrement.

CUSTOMER		ORD		ITEM	
NAME	CUSTID	CUSTID	ORDID	ORDID	ITEMID
JOCKSPORTS	100	101	610	610	3
TKB SPORT SHOP	101	102	611	611	1
VOLLYRITE	102	104	612	612	1
JUST TENNIS	103	106	601	601	1
K+T SPORTS	105	102	602	602	1
SHAPE UP	106	106	604
WOMENS SPORTS	107	106	605
...
9 rows selected.		21 rows selected.		64 rows selected.	

Figure 5 : la colonne CUSTID de la table CUSTOMER est reliée à la colonne CUSTID de la table ORD, la colonne ORDID de la table ORD est reliée à la colonne ORID de la table ITEM

Exemple :

```
SQL> SELECT      c.name, o.ordid, i.itemid, i.itemtot, o.total
2 FROM          customer c, ord o, item i
3 WHERE         c.custid = o.custid
4 AND           o.ordid = i.ordid
5 AND           c.name = 'TKB SPORT SHOP';
```

NAME	ORDID	ITEMID	ITEMTOT	TOTAL
TKB SPORT SHOP	610	3	58	101.4
TKB SPORT SHOP	610	1	35	101.4
TKB SPORT SHOP	610	2	8.4	101.4

3 rows selected.

Æ Cette requête affiche les orders, les numéros d'item, le total de chaque item et le total de chaque order pour le customer TKB SPORT SHOP.

La ligne 3 de la requête correspond à la jointure des tables CUSTOMER et ORDER.

La ligne 4 de la requête correspond à la jointure des tables ORDER et ITEM.

La ligne 5 est une condition qui vise à restreindre le résultat des deux jointures au customer portant le nom TKB SPORT SHOP.

1.3 Les autres types de jointures

1.3.1 Relier des tables avec une non équi-jointure

Une condition de non équi-jointure est utilisée lorsque deux tables n'ont pas de colonnes qui correspondent directement.

Il s'agit de la même syntaxe que pour la condition équi-jointure (cf 1.2.1).

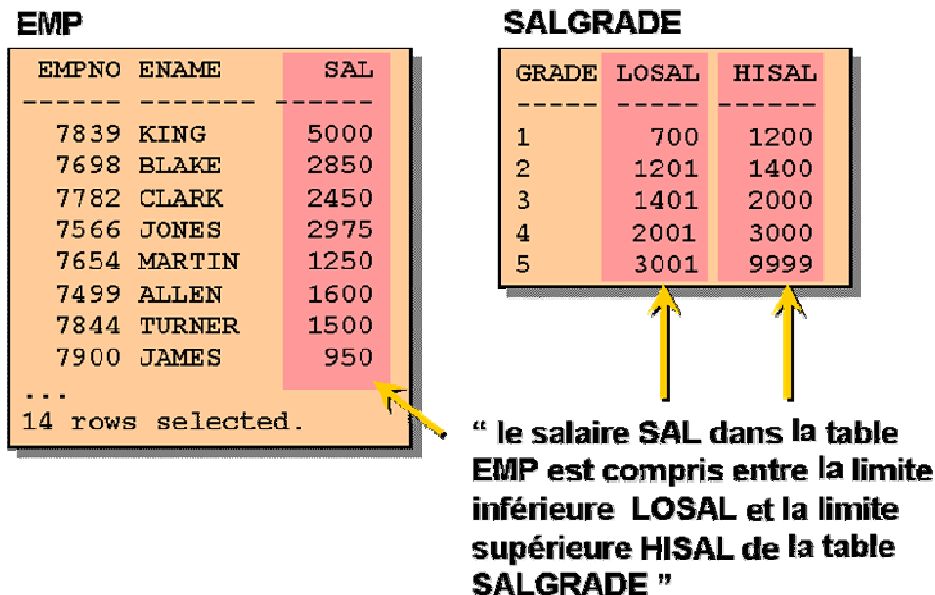


Figure 6 : Représentation d'une non equi-jointure

Exemple :

```
SQL> SELECT e.ename, e.sal, s.grade
2 FROM emp e, salgrade s
3 WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ENAME	SAL	GRADE
JAMES	950	1
SMITH	800	1
ADAMS	1100	1
...		

14 rows selected.

Æ Cette requête cherche la tranche de salaires de chaque employé. Or il n'existe pas de clé étrangère dans la table EMP faisant référence aux tranches de salaires de la table SALGRADE. Pour trouver la tranche de salaires de chaque employé, la requête va comparer les salaires des employés avec les limites de chaque tranche de salaires de la table SALGRADE. Cette relation est une non equi-jointure (ligne 3 de la requête).

Chaque employé n'apparaît qu'une seule fois dans le résultat de la requête. Il y a deux raisons à cela :

- Aucune limites des tranches de salaires dans la table SALGRADE ne se chevauchent. Donc le salaire d'un employé appartient au plus à une tranche.
- Aucun salaire n'est plus petit que la plus petite limite inférieure de tranche (700) et aucun salaire n'est plus grand que la plus grande limite supérieure de tranche (9999).

D'autres opérateurs tels que <= et >= peuvent être utilisés, mais l'opérateur BETWEEN, dans cet exemple, est plus simple à utiliser.

1.3.2 Relier des tables avec une jointure externe

Une condition de jointure externe (outer join) est utilisée pour afficher tous les enregistrements incluant ceux qui ne respectent pas la condition de jointure.

L'opérateur de jointure externe est le signe plus (+) :

```
SELECT      table1.column, table2.column
FROM table1, table2
WHERE       table1. column(+) = table2.column ;
```

Æ Cet requête affiche tous les enregistrements de la colonne de la table 1 même si ils ne respectent pas la condition de jointure

```
SELECT      table1.column, table2.column
FROM table1, table2
WHERE       table1. column = table2.column(+);
```

Æ Cette requête affiche tous les enregistrements de la colonne de la table 2 même si ils ne respectent pas la condition de jointure

L'opérateur de jointure externe ne peut apparaître que d'un seul côté de l'expression, le côté où il manque de l'information.

Une condition de jointure externe ne peut pas utiliser l'opérateur IN et ne peut pas être liée à une autre condition par l'opérateur OR.

Exemple :

EMP		DEPT	
ENAME	DEPTNO	DEPTNO	DNAME
-----	-----	-----	-----
KING	10	10	ACCOUNTING
BLAKE	30	20	RESEARCH
CLARK	10	30	SALES
JONES	30	40	OPERATIONS
...			
14 rows selected.		4 rows selected.	



Aucun employés dans le département OPERATIONS

Figure 7 : Exemple illustrant le besoin d'une condition de jointure externe

Utilisation d'une equi-jointure :

```
SQL> SELECT      e.ename, e.deptno, d.dname
2 FROM          emp e, dept d
3 WHERE         e.deptno = d.deptno;
```

```
ENAME      DEPTNO DNAME
-----
```

```

KING          10 ACCOUNTING
BLAKE         30 SALES
CLARK         10 ACCOUNTING
JONES         20 RESEARCH
...
ALLEN         30 SALES
TURNER        30 SALES
JAMES         30 SALES
...
14 rows selected.

```

Æ Cette requête affiche la liste des employés avec leur numéro et nom de département. Le département OPERATIONS n'apparaît dans le résultat. En effet, aucun employé n'y travaille.

Utilisation d'une jointure externe :

```

SQL> SELECT      e.ename, d.deptno, d.dname
  2 FROM          emp e,   dept d
  3 WHERE         e.deptno(+) = d.deptno
  4 ORDER BY     e.deptno;

ENAME          DEPTNO DNAME
-----
KING            10 ACCOUNTING
CLARK           10 ACCOUNTING
...
                40 OPERATIONS
15 rows selected.

```

Æ Cette requête affiche la liste des employés avec leur numéro et nom de département. Le département OPERATIONS apparaît cette fois dans le résultat malgré l'absence d'employé y travaillant.

1.3.3 Relier une table à elle-même avec une auto-jointure

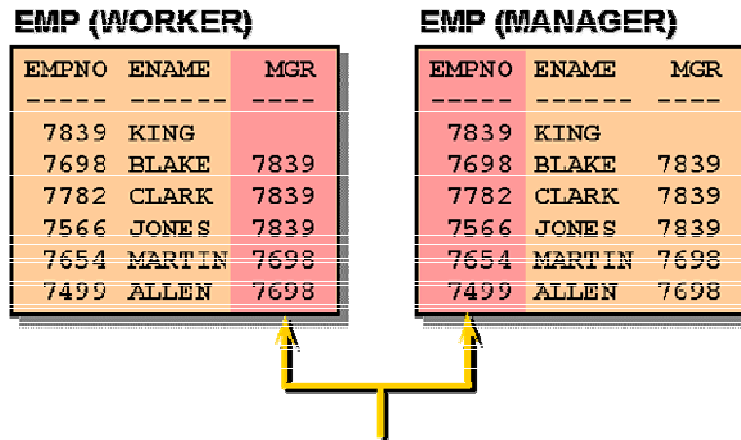
Une condition d'auto-jointure permet de faire une jointure sur deux colonnes liées appartenant à la même table.

```

SELECT          alias1.column, alias2.column
FROM            table1 alias1, table1 alias2
WHERE           alias1.column = alias2.column ;

```

Pour simuler deux tables dans la clause FROM, la table (*table1*) sur laquelle va être effectuée une auto-jointure va posséder deux alias (*table1 alias1, table1 alias2*).



“ MGR dans la table WORKER correspond à EMPNO dans la table MANAGER ”

Figure 8 : Représentation d’une auto-jointure

Exemple :

```
SQL> SELECT      worker.ename||' works for '||manager.ename reports
2  FROM          emp worker, emp manager
3  WHERE         worker.mgr = manager.empno ;

WORKER.ENAME||'WORKSFOR'||MANAG
-----
BLAKE works for KING
CLARK works for KING
JONES works for KING
MARTIN works for BLAKE
...
13 rows selected.
```

Æ Cette requête affiche la liste des employés avec le nom de leur manager.

2 LES FONCTIONS DE GROUPE

2.1 Les types de fonctions de groupe

2.1.1 Définition d'une fonction de groupe

Les fonctions de groupe sont utilisées pour afficher des informations sur un groupe d'enregistrements.

```
SELECT      [column,] group_function(argument)
FROM table
[WHERE      condition(s)]
[GROUP BY   column]
[ORDER BY   group_function(argument)] ;
```

argument peut-être un nom de colonne, une expression ou une constante.

Les fonctions de groupe ne peuvent pas être utilisées dans les clauses FROM, WHERE et GROUP BY.

2.1.2 Fonctions de groupe

SUM ([DISTINCT ALL] <i>n</i>)	Retourne la somme de toutes les valeurs du groupe <i>n</i>
MIN ([DISTINCT ALL] <i>expr</i>)	Retourne la plus petite valeur du groupe <i>expr</i>
MAX ([DISTINCT ALL] <i>expr</i>)	Retourne la plus grande valeur du groupe <i>expr</i>
COUNT ({ * [DISTINCT ALL] <i>expr</i> })	Retourne le nombre d'enregistrements contenus dans le groupe <i>expr</i> . COUNT(*) retourne le nombre total d'enregistrements retournés en incluant les valeurs nulles et les doublons.
AVG ([DISTINCT ALL] <i>n</i>)	Retourne la moyenne des valeurs du groupe <i>n</i>
STDDEV ([DISTINCT ALL] <i>x</i>)	Retourne la déviance standard de <i>x</i>
VARIANCE ([DISTINCT ALL] <i>x</i>)	Retourne la variance de <i>x</i>

Toutes ces fonctions de groupes ignorent les valeurs nulles sauf COUNT(*).

Le mot clé DISTINCT permet de ne pas prendre en compte les doublons.

Le mot clé ALL (par défaut) permet de prendre en compte toutes les valeurs incluant les doublons.

Le type de données des arguments peut être CHAR, VARCHAR2, NUMBER, DATE sauf pour les fonctions AVG, SUM, VARIANCE et STDDEV qui ne peuvent être utilisées qu'avec des données de type numérique.

Pour substituer les valeurs nulles dans un groupe, il faut utiliser la fonction single-row NVL (cf. cours SQLP " Module 1 : Ordres SELECT Basiques" paragraphe 4.6.1 " La fonction NVL ").

2.1.3 Utilisations des fonctions de groupe

Exemple 1 :

```
SQL> SELECT      AVG(sal), MAX(sal), MIN(sal), SUM(sal)
   2      FROM      emp
   3      WHERE      job LIKE 'SALES%';

AVG(SAL)  MAX(SAL)  MIN(SAL)  SUM(SAL)
-----
   1400    1600    1250    5600

1 row selected.
```

Æ Cette requête retourne la moyenne des salaires, le salaire maximum, le salaire minimum et la somme des salaires des employés dont la fonction commence par la chaîne de caractères « SALES ».

Exemple 2 :

```
SQL> SELECT      MIN(hiredate), MAX(hiredate)
   2  FROM      emp;

MIN(HIRED  MAX(HIRED
-----
17-DEC-80 12-JAN-83

1 row selected.
```

Æ Cette requête retourne la date d'embauche la plus récente et la date d'embauche la plus vieille.

Exemple 3 :

```
SQL> SELECT      COUNT(*)
   2  FROM      emp
   3  WHERE      deptno = 30;

COUNT(*)
-----
        6

1 row selected.
```

Æ Cette requête retourne le nombre d'enregistrements dans la table EMP dont la colonne DEPTNO a pour valeur 30.

Exemple 4 :

```
SQL> SELECT      COUNT(deptno)
   2  FROM      emp ;

COUNT(DEPTNO)
-----
        14
```

Æ Cette requête retourne le nombre de départements (doublons inclus) dans la table EMP.

Exemple 5 :

```
SQL> SELECT      COUNT(DISTINCT (deptno) )
   2  FROM      emp ;

COUNT(DISTINCT(DEPTNO) )
-----
        3
```

Æ Cette requête retourne le nombre de départements distincts dans la table EMP.

Exemple 6 :

```
SQL> SELECT AVG(comm)
      2 FROM emp;

AVG (COMM)
-----
      550
```

Æ Cette requête retourne la moyenne des commissions touchées par les employés. Le calcul de la moyenne ne tient pas compte des valeurs invalides telles que les valeurs nulles. Seulement quatre employés sont pris en compte dans le calcul, car ils sont les seuls à posséder une commission non nulle.

Exemple 7 :

```
SQL> SELECT AVG(NVL(comm,0))
      2 FROM emp;

AVG (NVL (COMM, 0))
-----
      157.14286
```

Æ Cette requête retourne la moyenne des commissions touchées par les employés en tenant compte des valeurs nulles. En effet, la fonction NVL substitue, le temps de la requête, les valeurs nulles par la valeur 0, ce qui permet de prendre les quatorze employés en compte pour le calcul de la moyenne.

2.2 Créer des groupes de données

2.2.1 La clause GROUP BY

La clause GROUP BY permet de diviser les enregistrements d'une table en groupes. Les fonctions de groupe peuvent être alors utilisées pour retourner les informations relatives à chaque groupe.

```
SELECT      [column1, ] group_function(column2)
FROM table_name
[WHERE      condition(s)]
[GROUP BY   column1]
[ORDER BY   column2];
```

Quelques règles :

- La clause WHERE peut être utilisée pour pré-exclure des enregistrements avant la division en groupes.
- Les colonnes de la clause FROM qui ne sont pas incluses dans une fonction de groupe doivent être présentes dans la clause GROUP BY.
- Les alias de colonne ne peuvent pas être utilisés dans la clause GROUP BY.
- Par défaut, la clause GROUP BY classe les enregistrements par ordre croissant. L'ordre peut être changé en utilisant la clause ORDER BY (cf. cours "Ordres SELECT Basiques" paragraphe 3.4.1 "La clause ORDER BY").

Exemple :

```
SQL> SELECT deptno, AVG(sal)
      2 FROM emp
      3 GROUP BY deptno;

DEPTNO  AVG (SAL)
```

```
-----
      10 2916.6667
      20   2175
      30 1566.6667

3 rows selected.
```

Æ Cette requête affiche la moyenne des salaires des employés pour chaque département présent dans la table EMP.

La colonne contenue dans la clause GROUP BY n'a pas obligatoirement besoin de se trouver dans la clause FROM.

La clause ORDER BY peut accueillir la ou les fonctions de groupe contenues dans la clause FROM.

Exemple :

```
SQL> SELECT deptno, AVG(sal)
2 FROM emp
3 GROUP BY deptno
4 ORDER BY AVG(sal) ;

DEPTNO  AVG(SAL)
-----
      30 1566.6667
      20   2175
      10 2916.6667

3 rows selected.
```

Æ Cette requête affiche la moyenne des salaires des employés pour chaque département présent dans la table EMP ordonné sur la moyenne.

2.2.2 Groupement sur plusieurs colonnes

Plusieurs colonnes peuvent être spécifiées dans la clause GROUP BY, ce qui permet de récupérer des informations d'un groupe intégré dans un autre groupe. (Organiser les données en sous-groupe).

```
SELECT      column1, column2, group_function(column)
FROM table
WHERE        condition(s)
GROUP BY    column1, column2
ORDER BY    column ;
```

Æ Les données seront organisées en groupes par rapport à la colonne column1. Puis chaque groupe sera à nouveau organisé en sous-groupes par rapport à la colonne column2.

EMP

DEPTNO	JOB	SAL
10	MANAGER	2450
10	PRESIDENT	5000
10	CLERK	1300
20	CLERK	800
20	CLERK	1100
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	SALESMAN	1600
30	MANAGER	2850
30	SALESMAN	1250
30	CLERK	950
30	SALESMAN	1500
30	SALESMAN	1250

“somme des salaires de la tables EMP pour chaque fonction , groupé par département”

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Figure 9 : Groupement sur plusieurs colonnes

Exemple :

```
SQL> SELECT deptno, job, sum(sal)
2 FROM emp
3 GROUP BY deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
...		

9 rows selected.

Æ Cette requête affiche la moyenne des salaires pour chaque fonction dans chaque département.

2.2.3 Restreindre le résultat des groupes avec la clause HAVING

La clause WHERE n'acceptant pas les fonctions de groupes, la restriction du résultat des fonctions de groupes se fera dans la clause HAVING.

```
SELECT      column, group_function(argument)
FROM table
[WHERE]      condition]
[GROUP BY]  group_by_expression]
[HAVING]     group_condition]
[ORDER BY]  column] ;
```

Comme dans les clauses WHERE et GROUP BY, les alias de colonne ne peuvent pas être utilisés dans la clause HAVING.

La clause HAVING peut être utilisée sans la présence de fonctions de groupe dans la clause FROM.

La différence entre HAVING et WHERE :

- WHERE restreint les enregistrements
- HAVING restreint les groupes d'enregistrements et peut-être utilisée pour restreindre les enregistrements.

Exemple 1 :

```
SQL> SELECT deptno, max(sal)
2 FROM emp
3 GROUP BY deptno
4 HAVING max(sal)>2900;
```

DEPTNO	MAX(SAL)
10	5000
20	3000

2 rows selected.

Æ Cette requête affiche les départements dont le salaire maximal dépasse \$2900.

Exemple 2 :

```
SQL> SELECT job, SUM(sal) PAYROLL
2 FROM emp
3 WHERE job NOT LIKE 'SALES%'
4 GROUP BY job
5 HAVING SUM(sal)>5000
6 ORDER BY SUM(sal);
```

JOB	PAYROLL
ANALYST	6000
MANAGER	8275

2 rows selected.

Æ Cette requête affiche la somme des salaires des employés supérieure à \$5000 pour chaque fonction dont les cinq premières lettres sont différentes de la chaîne de caractères « SALES ». Le résultat est ordonné de façon descendante sur les sommes des salaires.

2.2.4 Requêtes invalides utilisant des fonctions de groupe

Toutes les colonnes ou expressions dans la clause SELECT, qui ne sont pas le résultat d'une fonction de groupe, doivent être présentes dans la clause GROUP BY.

Exemple :

```
SQL> SELECT deptno, COUNT(ename)
2 FROM emp;

SELECT deptno, COUNT(ename)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

Un alias de colonne ne peut pas être utilisé dans le GROUP BY. Sinon l'erreur suivante se produit : " invalid column name ".

La clause HAVING ne peut pas être utilisée sans la clause GROUP BY.
Une fonction de groupe ne peut pas être utilisée dans la clause WHERE.

Exemple :

```
SQL> SELECT      deptno, AVG(sal)
  2 FROM          emp
  3 WHERE          AVG(sal) > 2000
  4 GROUP BY      deptno;

WHERE AVG(sal) > 2000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

2.2.5 Les fonctions de groupe imbriquées

Des fonctions de groupe ayant comme argument le résultat d'une fonction de groupe sont appelées fonctions imbriquées (nesting functions).

Exemple :

```
SQL> SELECT      MAX(AVG(sal))
  2 FROM          emp
  3 GROUP BY      deptno;

MAX(AVG(SAL))
-----
      2916.6667
```

Æ AVG(sal) calcule la moyenne des salaires dans chaque département grâce à la clause GROUP BY. MAX(AVG(sal)) retourne la moyenne des salaires la plus élevée.

3 LES SOUS-REQUETES

3.1 Les sous-requêtes basiques

3.1.1 Les règles de conduite des sous-requêtes

Une sous-requête est une clause SELECT imbriquée dans une clause d'un autre ordre SQL. Une sous-requête peut être utile lorsqu'il faut sélectionner des enregistrements en utilisant une condition qui dépend d'une valeur inconnue d'une autre colonne.

Exemple :

L'objectif est d'écrire une requête qui identifie tous les employés qui touchent un salaire plus grand que celui de l'employé Jones, mais la valeur du salaire de cet employé n'est pas connu. Dans ce cas, il faut faire appel à une sous-requête qui va retourner le salaire de Jones à la requête principale.

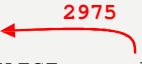
Pour combiner deux requêtes, il suffira de placer une requête à l'intérieur d'une autre. La requête à l'intérieure (ou la sous-requête) retourne une valeur qui est utilisée par la requête extérieure (ou requête principale). L'utilisation d'une sous-requête est équivalente à l'utilisation de deux requêtes séquentielles. Le résultat de la première requête est la valeur recherchée dans la seconde requête.

SELECT *select_list*
FROM *table*
WHERE *expression operator (*

SELECT *select_list*
FROM *table) ;*

Exemple 1 :

```
SQL> SELECT      ename
2 FROM          emp
3 WHERE         sal > (
                  SELECT      sal
                  FROM        emp
                  WHERE       ename = 'JONES' ) ;
```



Æ La sous-requête retourne le salaire de l'employé JONES. La requête principale compare le salaire des employés au salaire de l'employé JONES et ne retourne que ceux qui lui sont supérieurs.

Règles de conduite :

- Une sous-requête doit être mise entre parenthèses.
- Une sous-requête doit être placée du côté droit de l'opérateur de comparaison.
- Une sous-requête ne possède pas de clause ORDER BY.
- Une sous-requête peut être seulement placée dans les clauses WHERE, HAVING et FROM.

3.1.2 Les types de sous-requête

Il existe trois types de sous-requête :

- Single-row : Retourne une valeur contenue dans une colonne.
- Multiple-row : Retourne plusieurs valeurs contenues dans une colonne.
- Multiple-column : Retourne les valeurs de plusieurs colonnes

- **Single-row subquery**



- **Multiple row subquery**



- **Multiple column subquery**



Figure 10 : Les types de sous-requêtes (subqueries)

3.2 Les sous-requêtes Single-Row

3.2.1 Ecriture d'une sous-requête single-row

Une sous-requête single-row se situe dans la clause WHERE de la requête principale.

Une sous-requête single-row ne peut être utilisée qu'avec les opérateurs de comparaison suivants : <, >, =, <=, >=, <>. (cf. cours SQLP " Module 1 : Ordres SELECT Basiques" paragraphe 3.2.1 " Les opérateurs de comparaisons ").

Exemple :

```
SQL> SELECT      ename, job
2  FROM          emp
3  WHERE         job =
4                ( SELECT      job
5                  FROM        emp
6                  WHERE       empno = 7369 );
```

ENAME	JOB
JAMES	
	CLERK
SMITH	CLERK
ADAMS	CLERK
MILLER	CLERK

4 rows selected.

Æ Cette requête affiche les employés dont la fonction est la même que celle de l'employé numéro 7369.

Plusieurs sous-requêtes peuvent être mises en place dans une requête.

Exemple de syntaxe :

SELECT *select_list*
FROM *table*
WHERE *expression single-row_comparison_operator* (

AND *expression single-row_comparison_operator* (

SELECT *select_list*
FROM *table*)

SELECT *select_list*
FROM *table*) ;

Exemple :

```
SQL> SELECT      ename, job
2 FROM          emp
3 WHERE         job =
4                ( SELECT      job
5                  FROM        emp
6                  WHERE       empno = 7369)
7
8 AND          sal >
9                ( SELECT      sal
10                 FROM        emp
11                 WHERE       empno = 7876);
```

CLERK
1100

ENAME	JOB
MILLER	CLERK

1 row selected.

Æ Cette requête affiche la liste des employés dont la fonction est la même que l'employé numéro 7369 et dont le salaire est le même que l'employé 7876.

3.2.2 Utilisation de fonctions de groupe dans des sous-requêtes single-row

Des fonctions de groupes peuvent être utilisées dans une sous-requête pour opérer sur un groupe de valeurs. (cf. dans ce cours paragraphe 2 "Les fonctions de groupe").

Placer dans la sous-requête, elles permettent de ne retourner qu'une seule valeur à la requête principale.

Exemple :

```
SQL> SELECT      ename, job, sal
2 FROM          emp
3 WHERE         sal =
4                ( SELECT      MIN(sal)
5                  FROM        emp);
```

800

ENAME	JOB	SAL
SMITH	CLERK	800

1 row selected.


Æ Cette requête affiche le nom, la fonction et le salaire des employés dont le salaire est égal au salaire minimum. La fonction MIN ne retourne qu'une seule valeur (800).

3.2.3 Des sous-requêtes dans la clause HAVING

La clause HAVING peut contenir une sous-requête et des fonctions de groupe.

Exemple 1 :

```
SQL> SELECT deptno, MIN(sal)
2 FROM emp
3 GROUP BY deptno
4 HAVING MIN(sal) >
5 ( SELECT MIN(sal)
6 FROM emp
7 WHERE deptno = 20);
```



DEPTNO	MIN(SAL)
10	1300
30	950

2 rows selected.

Æ Cette requête affiche les départements qui ont un salaire minimum plus grand que le salaire minimum du département 20.

Exemple 2 :

```
SQL > SELECT job, AVG(sal)
2 FROM emp
3 GROUP BY job
4 HAVING AVG(sal) = ( SELECT MIN(AVG(sal))
5 FROM emp
6 GROUP BY job) ;
```

Æ Cette requête affiche la fonction qui a le salaire moyen le plus bas. La sous-requête ne retourne qu'un seul enregistrement grâce aux fonctions de groupe dans la clause SELECT de la sous-requête.

3.2.4 Les problèmes les plus courants lors de l'utilisation de sous-requêtes single-row

Si la sous-requête contient la clause GROUP BY, cela signifie qu'elle retourne plusieurs enregistrements (lignes). Il faut s'assurer que la sous-requête single-row ne retournera qu'un seul enregistrement.

Exemple :

```
SQL> SELECT empno, ename
2 FROM emp
3 WHERE sal = ( SELECT MIN(sal)
4 FROM emp
5 GROUP BY deptno ) ;
```

ERROR:
ORA-01427: single-row subquery returns more than one row

no rows selected

Æ La sous-requête retourne trois valeurs : 800, 1300 et 950. La clause WHERE contient l'opérateur de comparaison = single-row n'acceptant qu'une seule valeur. L'opérateur ne peut pas accepter plus d'une valeur provenant de la sous-requête, c'est pourquoi il génère une erreur. Pour corriger la requête, il suffit de changer l'opérateur = par l'opérateur IN.

Un problème courant avec les sous-requêtes est lorsque aucune valeur n'est retournée par la sous-requête.

Exemple :

```
SQL> SELECT ename, job
2 FROM emp
```

```

3  WHERE      job =
4              (SELECT    job
5                   FROM    emp
6                   WHERE   ename='SMYTHE' );

no rows selected

```

Æ Cette requête n'affiche aucun résultat. Aucun n'employé ne s'appelle SMYTHE, donc la sous-requête ne retourne aucune valeur. La requête principale compare le résultat de la sous-requête (null) dans sa clause WHERE. La requête principale ne trouve aucun employé dont la fonction est égale à null et ne retourne donc aucun enregistrement.

3.3 Les sous-requêtes Multiple-Row

3.3.1 Règles de conduite des sous-requêtes multiple-row

Une sous-requête multiple-row retourne une liste de valeurs qui seront comparées à une seule valeur dans la requête principale.

Une sous-requête multiple-row ne peut être utilisée qu'avec les opérateurs de comparaison multiple-row : IN, NOT IN, ANY, ALL, BETWEEN. (cf. cours SQLP " Module 1 : Ordres SELECT Basiques" paragraphe 3.2 " Les opérateurs de comparaisons ").

Une sous-requête multiple-row ne peut pas contenir une clause ORDER BY. En effet, l'ordonnancement du résultat de la sous-requête n'est pas nécessaire : la requête principale n'utilise pas d'index quand elle compare la valeur avec chaque résultat de la sous-requête.

Les fonctions de groupe peuvent être utilisées dans une sous-requête multiple-row.

Exemple :

```

SQL> SELECT      ename, sal, deptno
2  FROM          emp
3  WHERE         sal IN (
4                  SELECT      MIN(sal)
5                  FROM        emp
6                  GROUP BY    deptno);

```

800, 950, 1300

Æ Cette requête affiche les employés dont le salaire est égal au salaire minimum d'un département. Elle est équivalente à la requête suivante :

```

SQL> SELECT      ename, sal, deptno
2  FROM          emp
3  WHERE         sal IN (800, 950, 1300);

```

3.3.2 Ecriture de sous-requêtes multiple-row

Une sous-requête multiple-row est utilisée pour récupérer un ensemble de valeurs qui sera comparé à la valeur dans la clause WHERE de la requête principale.

Exemple 1 :

```

SQL> SELECT      ename, mgr
2  FROM          emp
3  WHERE         mgr IN ( SELECT      empno
4                          FROM        emp
5                          WHERE       deptno = 20 );

```

Æ Cette requête affiche les employés dont le manager travaille dans le département n°20.

Exemple 2 :


```

SQL> SELECT      empno, ename, job
2 FROM          emp
3 WHERE         sal < ANY
4               ( SELECT sal
5                 FROM emp
6                 WHERE job = 'CLERK')
7 AND          job <> 'CLERK';

```

1300, 800, 950

EMPNO	ENAME	JOB
7654	MARTIN	SALESMAN
7521	WARD	SALESMAN

2 rows selected.

Æ Cette requête affiche les employés dont la fonction n'est pas CLERK et dont le salaire est plus petit que les employés dont la fonction est CLERK. Le salaire maximum qu'un employé dont la fonction est CLERK est de \$1300. La requête affiche tous les employés dont la fonction n'est pas CLERK et dont le salaire est inférieur à \$1300.

L'opérateur ANY (équivalent à l'opérateur SOME) compare une valeur à chaque valeur retournée par la sous-requête.

< ANY signifie plus petit que le minimum.

> ANY signifie plus grand que le maximum.

= ANY est équivalent à IN.

L'opérateur NOT ne peut être utilisé avec l'opérateur ANY.

Exemple 3 :

```

SQL> SELECT      empno, ename, job
2 FROM          emp
3 WHERE         sal > ALL
4               ( SELECT      AVG(sal)
5                 FROM        emp
6                 GROUP BY    deptno);

```

EMPNO	ENAME	JOB
7839	KING	PRESIDENT
7566	JONES	MANAGER
7902	FORD	ANALYST
7788	SCOTT	ANALYST

4 rows selected.

Æ Cette requête affiche les employés dont le salaire est plus grand que la moyenne des salaires de chaque département.. La plus grande moyenne des salaires pour chaque département est de \$2916,66. La requête principale retourne donc les employés dont le salaire est supérieur à \$2916,66.

L'opérateur ALL compare une valeur à toutes les valeurs retournées par la sous-requête.

> ALL signifie plus grand que le maximum.

< ALL signifie plus petit que le minimum.

L'opérateur NOT ne peut être utilisé avec l'opérateur ALL.

3.3.3 Les valeurs nulles dans les sous-requêtes multiple-row

Si la sous-requête retourne une valeur nulle à la requête principale, la requête principale ne retournera pas d'enregistrements. Pour palier à ce problème, il faut utiliser la fonction NVL.

3.4 Les sous-requêtes Multiple-column

3.4.1 Ecritures de sous-requêtes multiple-column

Pour comparer deux colonnes ou plus dans une clause WHERE, il faut utiliser les opérateurs logiques.

Les colonnes spécifiées dans la requête principale doivent correspondre aux colonnes dans la sous-requête : il doit y avoir le même nombre de colonne et les types de données doivent correspondre.

Si il n'y a pas le même nombre de colonne, il y aura une erreur, si les types ne correspondent pas, les résultats risquent d'être faux.

```
SELECT      column1, column2
FROM table1
WHERE      (column1, column2) IN (      SELECT      column3, column4
                                      FROM table2
                                      WHERE      condition ) ;
```

La clause WHERE de la requête principale attend des valeurs issues de deux colonnes qui seront comparées aux valeurs de *column1* et *column2*.

La sous-requête retourne les valeurs issues de deux colonnes *column3* et *column4*.

column1 et *column3* doit être du même type de données.

column2 et *column4* doit être du même type de données.

Exemple :

```
SQL> SELECT      ename, job, hiredate
2 FROM          emp
3 WHERE         (hiredate, job) IN ( SELECT      MIN(hiredate), job
4                                     FROM          emp
5                                     GROUP BY      job );
```

ENAME	JOB	HIREDATE
SMITH	CLERK	17/12/80
ALLEN	SALESMAN	20/02/81
JONES	MANAGER	02/04/81
KING	PRESIDENT	17/11/81
FORD	ANALYST	03/12/81

5 rows selected.

Æ Cette requête affiche les employés possédant la date d'embauche la plus récente pour chaque fonction.

3.4.2 Comparaisons de plusieurs colonnes

Une comparaison "pairwise" peut générer des résultats différents d'une comparaison non "pairwise".

Exemple d'une comparaison "pairwise" :

```
SQL> SELECT      ename, deptno, sal, comm
2 FROM          emp
3 WHERE         (sal, NVL(comm,-1)) IN
4                                     ( SELECT      sal, NVL(comm,-1)
5                                     FROM          emp
6                                     WHERE         deptno = 30 )
7 AND          deptno <> 30;
```

no rows selected.

Æ Cette requête affiche les employés qui ont le même salaire et la même commission qu'un des employés du département 30, mais qui ne font pas parti du département 30. La sous-requête retourne la liste des salaires et des commissions des employés du département 30. Les valeurs fonctionnent par couple : un salaire et une commission. La requête principale compare les couples salaire/commission de la table des employés avec les couples retournés par la sous-requête et retourne ceux qui correspondent. Aucun n'employé ne faisant pas parti du département 30 n'a le même salaire et la même commission qu'un employé du département 30.

Exemple d'une comparaison non "pairwise" :

```
SQL> SELECT ename, deptno, sal, comm
2  FROM emp
3  WHERE sal IN
4      ( SELECT sal
5        FROM emp
6        WHERE deptno = 30 )
7  AND NVL(comm,-1) IN
8      ( SELECT NVL(comm,-1)
9        FROM emp
10       WHERE deptno = 30 )
11 AND deptno <> 30 ;
```

ENAME	DEPTNO	SAL	COMM
CLARK	10	1500	300

1 rows selected.

Æ La condition multiple-column de la première requête "pairwise" a été divisée en deux conditions multiple-row reliées par l'opérateur AND. La nouvelle requête n'aboutit plus au même résultat. Elle affiche les employés qui ne font pas parti du département 30, qui ont le même salaire qu'un employé du département 30 et la même commission qu'un employé du département 30. La requête ne retourne qu'un seul enregistrement : CLARK. En effet CLARK a le même salaire que TURNER et la même commission que ALLEN.

3.4.3 L'utilisation de sous-requêtes multiple-column dans la clause FROM

Des sous-requêtes multiple-column peuvent être écrites dans la clause FROM. La méthode est similaire quelque soit le type de sous-requête.

Le résultat de la sous-requête dans la clause FROM est une table virtuelle. Cette table virtuelle doit posséder un alias de table afin d'identifier le résultat de la sous-requête.

Exemple :

```
SQL> SELECT      a.ename, a.sal, a.deptno, b.salavg
2  FROM          emp a, (
3                SELECT      deptno, AVG(sal) salavg
4                FROM        emp
5                GROUP BY    deptno ) b
6  WHERE         a.deptno = b.deptno
7  AND          a.sal < b.salavg ;
```

ENAME	SAL	DEPTNO	SALAVG
CLARK	2450	10	4687,25

MILLER	1300	10	4687,25
SMITH	800	20	2175
ADAMS	1100	20	2175
MARTIN	1250	30	1566,66667
JAMES	950	30	1566,66667
TURNER	1500	30	1566,66667
WARD	1250	30	1566,66667
8 rows selected.			

Æ Cette requête affiche, pour chaque employé touchant un salaire inférieur à la moyenne des salaires de son département, son nom, son salaire, son numéro de département et la moyenne des salaires dans son département.

La sous-requête retourne les numéros de département et la moyenne des salaires pour chaque département. Ces résultats sont stockés dans une table virtuelle appelée *b*.

Une jointure) relie la table *emp* et la table virtuelle *b* (ligne 6).

Le salaire de chaque employé est ensuite comparé au salaire moyen de son département (*salavg*) obtenu dans la sous-requête.

4 CREER DES RAPPORTS AVEC SQL*PLUS

4.1 Les variables de substitution

4.1.1 Caractéristiques des 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 active SQL*Plus 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 '&'.

Une variable de substitution peut être placée d'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.

Syntaxe d'une variable de substitution :

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

Exemple d'utilisation :

- Les variables de substitution peuvent être utilisées pour demander à l'utilisateur de saisir un mot de passe.
- Les entêtes et pieds de page de l'état peuvent être modifiés dynamiquement. Les états peuvent s'afficher différemment suivant les utilisateurs.

A l'aide de variable de substitution, SQL*Plus peut être utilisé pour créer des états interactifs dans un fichier script ou dans un ordre SQL.

4.1.2 Substitution de nombres

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.

Exemple:

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

Entrez une valeur pour department : 10

DNAME                DEPTNO
-----
ACCOUNTING            10

1 row selected.
```

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

La commande **SET VERIFY** active SQL*Plus à fournir un feedback (retour de l'information) à l'utilisateur à propos des variables de substitution et des valeurs qui ont été saisies précédemment.

SET VERIFY { ON | OFF }

Par défaut la commande SET VERIFY est active (ON).

Exemple :

```
SQL> SET VERIFY ON ;

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

Entrez une valeur pour department : 10

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

DNAME                                DEPTNO
-----
ACCOUNTING                            10

1 row selected.
```

Æ Après la saisie de l'utilisateur, SQL*Plus affiche la mise à jour de la ligne dans le code SQL, et seulement après le résultat de la requête.

4.1.3 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.

Syntaxe d'une variable substituant une valeur de type date ou chaîne de caractères : '&user_variable'

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

2 rows selected.
```

Æ Cette requête affiche le nom, le département et le salaire annuel des employés dont le fonction 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.

4.1.4 Utilisations des variables de substitution

Une variable de substitution peut aussi substituer un nom de colonne, une expression, un nom de table, la liste des éléments dans une clause SELECT, une expression, la liste des éléments d'ordonnancement dans une clause ORDER BY, une condition dans la clause WHERE ou encore du texte.

Exemples :

```
SQL> SELECT      ename, &column_name
2 FROM          emp ;
```

Æ La variable *column_name* substitue un nom de colonne dans la clause SELECT.

```
SQL> SELECT      empno, deptno
2 FROM          emp
3 WHERE         deptno = 10
4 ORDER BY     &order_column ;
```

Æ La variable *order_column* substitue un nom de colonne dans la clause ORDER BY.

```
SQL> SELECT      empno, job, &column_name
2 FROM          emp
3 WHERE         &condition ;
```

Æ Cette requête combine les deux variables de substitution des exemples précédents.

```
SQL> SELECT      ename, empno, job, &column_name
2 FROM          emp
3 WHERE         &condition
4 ORDER BY     &order_column ;
```

Æ Cette requête combine les deux variables de substitution des exemples précédents avec la variable *condition* qui substitue une condition dans la clause WHERE.

```
SQL> SELECT      *
2 FROM          &table_name ;
```

Æ La variable *table_name* substitue le nom d'une table dans la clause FROM.

4.1.5 Variable de substitution avec un 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	Æ pour les valeurs de type numérique
'&&user-variable'	Æ 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 ;
```

Enter value for column_name: deptno

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7782	CLARK	MANAGER	10
7934	MILLER	CLERK	10
...			

14 rows selected.

Æ 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.

4.2 Les variables définies par l'utilisateur

Des variables peuvent être définies avant l'exécution d'un ordre SQL.

SQL*Plus fournit deux commandes pour définir et initialiser des variables : **DEFINE** et **ACCEPT**.

4.2.1 La commande ACCEPT

La commande ACCEPT lit la valeur saisie par l'utilisateur et la stocke dans une variable.

ACCEPT *user_variable* [*datatype*] [**FOR**[**MAT**] *format*] [**PROMPT** *text*] [**HIDE**]

<i>user_variable</i>	Le nom de la variable ne doit pas être précédé d'un '&' dans la commande ACCEPT.
<i>datatype</i>	Le type de données est un paramètre optionnel qui peut-être : <ul style="list-style-type: none"> - NUMBER : la variable sera convertie en type de données NUMBER - CHAR : la variable sera convertie en CHAR, la taille maximale d'une variable CHAR est de 240 bytes - DATE : la variable sera convertie en un format de date valide qui est DD-MON-YY.
FOR [MAT] <i>format</i>	FORMAT est un paramètre optionnel qui permet de spécifier un model de format. Le format est un paramètre optionnel. Exemple : FORMAT \$9,999.00
PROMPT <i>text</i>	PROMPT est un paramètre optionnel qui permet d'afficher un message sur la ligne où l'utilisateur saisit sa valeur. Si le message contient un espace ou un signe de ponctuation, il doit être placé entre simples côtes. Si aucun message n'est spécifié, une ligne blanche sera affichée.
HIDE	HIDE est un paramètre optionnel qui permet de ne pas afficher le texte que l'utilisateur va saisir. Exemple : SQL> Enter password : *****

Si la variable spécifié n'existe pas, SQL*Plus l'a créera.

Si la commande ACCEPT est tapée en ligne de commande, la valeur de la variable sera immédiatement demandée.

Le nom de la variable ne doit pas être précédé d'un "&" dans la commande ACCEPT.

Exemple :

```
SQL> ACCEPT dept PROMPT 'Provide the department name : '
Provide the department name : Sales

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

DEPTNO DNAME LOC
-----
30 SALES CHICAGO

1 row selected.
```

Æ SQL*Plus demande la saisie de la valeur de la variable *dept* au moment de l'exécution de la commande ACCEPT et l'utilise ensuite lors de l'exécution de l'ordre SELECT.

Exemple :

```
SQL> ACCEPT user_pass PROMPT 'Enter your password : ' HIDE
Enter your password : *****
```

Æ Cette commande ACCEPT permet de cacher la valeur saisie par l'utilisateur grâce au paramètre HIDE.

4.2.2 Les Commandes DEFINE et UNDEFINE

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

La commande **UNDEFINE** est utilisée pour effacer les variables. A la fermeture d'une session, toutes les variables définies au cours de cette session sont effacées. Pour éviter cela, le fichier login.sql peut être modifié pour que les variables soit recréées au démarrage (cf. dans ce cours paragraphe 4.3.1 "Les variables systèmes et la commande SET").

DEFINE <i>variable</i> = <i>value</i>	Crée la variable utilisateur <i>variable</i> de type CHAR et lui assigne la valeur <i>value</i> .
DEFINE <i>variable</i>	Affiche la variable <i>variable</i> , sa valeur et son type de données.
DEFINE	Affiche toutes les variables utilisateurs, leur valeur et leur type de données.
UNDEFINE <i>variable</i>	Efface la variable <i>variable</i> .

Exemple 1 :

```
SQL> DEFINE deptname = sales
SQL> DEFINE deptname

DEFINE DEPTNAME          = "sales" (CHAR)
```

Æ La première commande DEFINE définit la variable *deptname* et lui attribue la valeur *sale*.

La deuxième commande DEFINE affiche la variable *deptname*, sa valeur (*sales*) et son type de données (CHAR).

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

DEPTNO    DNAME      LOC
-----
      30 SALES      CHICAGO

1 row selected.
```

Æ Cette requête affiche le département dont le nom est défini par la variable *deptname* dont la valeur est *sale*. La variable *deptname* définie par la commande DEFINE s'utilise comme n'importe quelle variable.

Exemple 2 :

```
SQL> UNDEFINE deptname
SQL> DEFINE deptname
symbol deptname is UNDEFINED
```

Æ La commande UNDEFINE efface la variable *deptname*. La commande DEFINE demande la définition de la variable *deptname*. Comme *deptname* n'existe plus, SQL*Plus est incapable de donner sa définition et en informe l'utilisateur avec le message ci-dessus.

4.3 Personnaliser l'environnement SQL*Plus

4.3.1 Les variables système et la commande SET

La commande **SET** sert à contrôler l'environnement SQL*Plus.

SQL> **SET** system_variable value

On peut utiliser la commande **SHOW** pour afficher le statut courant d'une variable système.

Exemple :

```
SQL> SHOW ECHO
echo off
```

On peut utiliser la commande **SHOW ALL** pour afficher le statut courant de toutes les variables systèmes.

Les huit variables systèmes basiques :

COLSEP	{ <u>_</u> <i>text</i> }	Afficher la chaîne de caractères <i>text</i> ou des blancs (part défaut) entre les colonnes
HEA[DING]	{ OFF ON }	Afficher les entêtes de colonnes
FEED[BACK]	{ <u>6</u> <i>n</i> OFF ON }	Nombre d'enregistrements retournés par la requête à partir duquel SQL*Plus affiche le message des enregistrements sélectionnés
LIN[ESIZE]	{ <u>80</u> <i>n</i> }	Nombre de caractères par ligne en sortie
PAGES[IZE]	{ <u>24</u> <i>n</i> }	Nombre de lignes par page en sortie
LONG	{ <u>80</u> <i>n</i> }	Longueur maximal lors de l'affichage d'un LONG

PAU[SE] { <u>OFF</u> ON <i>text</i> }	Effectuer une pause à chaque début de page
TERM[OUT] { OFF <u>ON</u> }	Afficher les résultats de la requête
ARRAY[SIZE] { <u>20</u> <i>n</i> }	Définir la taille « database data fetch »

La valeur soulignée est la valeur par défaut.

Lors de la fermeture d'une session, tous les modifications effectuées sur les variables système sont perdus. Pour éviter de taper à chaque ouverture de session les changements de variables, il suffit de créer un fichier **LOGIN.SQL** qui contiendra les commandes SET. A chaque ouverture de session, SQL*Plus chargera le fichier et l'exécutera. LOGIN.SQL peut aussi contenir d'autres commandes variées qui n'ont pas été décrites ci-dessus.

4.3.2 Les commandes de formatage SQL*Plus

SQL*Plus fournit des commandes de formatage qui permettent de configurer les formatages des états :

COL[UMN] [<i>column option</i>]	Contrôle le format des colonnes
BRE[AK] [ON <i>report-element</i>]	Supprime les doublons et divise les enregistrements en section
TTI[TLE] [<i>text</i> OFF ON]	Spécifie l'entête de chaque page de l'état
BTI[TLE] [<i>text</i> OFF ON]	Spécifie le pied de chaque page de l'état

Les changements appliqués aux variables de formatages sont valables jusqu'à la fin de la session. Si un alias de colonne est utilisé, les commandes de formatage se référeront à l'alias de colonne et non au nom de la colonne elle-même. Les paramètres reprendront leur valeur par défaut après l'édition de chaque rapport (après l'exécution de chaque requête).

4.3.3 La commande COLUMN

La commande COLUMN contrôle l'affichage d'une colonne.

COL[UMN] [{ *column* | *alias* } [*option*]]

Les options de la commande COLUMN :

CLE[AR] <i>column_name</i>	Effacer tous les formats de la colonne <i>column_name</i> .
FOR[MAT] <i>format</i>	Formater une colonne avec les models de formatage de la commande FORMAT (cf. cours SQLP "Module 1 : Ordres SELECT Basiques" paragraphe 4.5.3 "La fonction TO_CHAR avec des nombres").
HEA[DING] <i>text</i>	Affecter un entête de colonne (mettre des simples côtes si le texte contient des espaces ou des signes de ponctuation, la ligne verticale spécifie un retour chariot)
JUS[TIFY] { <i>align</i> }	Justifier l'entête d'une colonne : R[IGHT], L[EFT] ou C[ENTER]
NOPRI[NT] PRI[NT]	Afficher ou pas une colonne
NUL[L] { <i>text</i> }	Afficher la chaîne de caractères <i>text</i> à la place des valeurs nulles

TRU[NCATED]	Tronquer une chaîne de type CHAR, VARCHAR2, LONG ou DATE qui est trop grande pour une colonne
WRA[PPED]	Mettre les valeurs de la colonne sur deux lignes quand la chaîne est trop grande pour tenir sur la colonne.

La commande COLUMN n'affecte pas les données dans la base de données.

Afficher ou effacer le paramétrage des commandes :

COL[UMN] <i>column_name</i>	Afficher le paramétrage de la colonne <i>column_name</i>
COL[UMN]	Afficher le paramétrage de toutes les colonnes
COL[UMN] <i>column_name</i> CLE[AR]	Effacer le paramétrage de la colonne <i>column_name</i>
CLE[AR] COL[UMN]	Effacer le paramétrage de toutes les colonnes

Si la commande est trop longue et qu'elle doit continuer sur une nouvelle ligne, la ligne courante doit se terminer par (-) avant de passer à la ligne suivante.

4.3.4 La commande BREAK

La commande **BREAK** place un espace entre les enregistrements, supprime les doublons pour une colonne donnée, saute une ligne à chaque fois qu'une valeur d'une colonne donnée change et spécifie l'endroit où imprimer.

La commande BREAK sert à clarifier et organiser les états.

BRE[AK] [ON *column* [*action*]]

element peut être le nom d'une colonne, ou peut être le mot clé REPORT.
action définit l'action du break :

SKIP <i>n</i>	Saute <i>n</i> lignes entre deux enregistrements quand le BREAK intervient
PAGE	Saute à une nouvelle page quand le BREAK intervient (réaffichage des entêtes de colonne pour chaque groupe)
DUP[LICATES]	Affiche les doublons

On peut spécifier plusieurs clauses ON dans une commande BREAK.

REPORT crée un groupe au niveau de l'état. La clause **BREAK ON REPORT** spécifie une position dans l'état où SQL*Plus positionnera les valeurs 'grand computed'.

La commande **CLEAR BREAKS** efface tous les paramètres BREAK.

Exemple :

```
SQL> BREAK ON job
SQL> SELECT job, ename, sal
2 FROM emp
3 ORDER BY job ;
```

JOB	ENAME	SAL
ANALYST	SCOTT	3000
	FORD	3000
CLERK	SMITH	800
	ADAMS	1100

	MILLER	1300
	JAMES	950
MANAGER	JONES	2975
	CLARK	2450
	BLAKE	2850
PRESIDENT	KING	5000
SALESMAN	ALLEN	1600
	MARTIN	1250
	TURNER	1500
	WARD	1250
14 rows selected.		

Æ Cette requête affiche la liste des employés ordonnés suivant leur fonction. La commande **BREAK** permet d'afficher qu'une seule fois chaque fonction, ce qui clarifie le rapport (le résultat de la requête).

Pour utiliser des **BREAK** dans une requête, cette dernière doit posséder une clause **ORDER BY**.

4.3.5 Les commandes TITLE

La commande de formatage **TTITLE** permet d'afficher des informations dans la section d'entête de chaque page de l'état.

TTITLE[TLE] [*text* | *variable*] [**OFF** | **ON**]

La commande de formatage **TTITLE** permet d'afficher des informations dans la section de pied de chaque page de l'état.

BTITLE[TLE] [*text* | *variable*] [**OFF** | **ON**]

Le paramètre *text* représente le texte qui apparaîtra dans la section concernée. Si le texte contient des espaces ou des signes de ponctuation, il doit être entouré de simples côtes. Utiliser le caractère "|" pour effectuer un retour à la ligne dans votre section.

Les paramètres par défaut sont :

- le texte au centre
- la date dans le coin gauche
- le nombre de page dans le coin droit

Le paramètre **PRINTSEC** dans une commande **TITLE** spécifie les valeurs des paramètres de formatage pour personnaliser les sections de l'état.

Options du paramètre **PRINTSEC** :

COL	<i>n</i>	
S[KIP]	<i>n</i>	Saute <i>n</i> lignes entre le titre et les colonnes
TAB	<i>n</i>	Tabulation de <i>n</i> position (peut y en avoir plusieurs dans le même ordre SQL*Plus)
LE[FT]	<i>text</i>	Place la chaîne de caractères <i>text</i> à gauche
CE[NTER]	<i>text</i>	Place la chaîne de caractères <i>text</i> au centre
R[IGHT]	<i>text</i>	Place la chaîne de caractères <i>text</i> à droite
BOLD	<i>text</i>	Met la chaîne de caractères <i>text</i> en gras (SQL*Plus représente le texte en gras en l'écrivant trois fois à la suite)
FORMAT	<i>text</i>	Formate la chaîne de caractères <i>text</i>

La variable **SQLPNO** stocke le numéro de page courant.

Pour afficher la configuration courante des titres, il suffit de taper :

```
SQL> TTITLE
SQL> BTITLE
```

TTITLE inclus automatiquement la date et le nombre de page au rapport.

Exemple :

```
SQL> TTITLE 'Job|Report '
SQL> BTITLE 'Confidential'
SQL> SELECT      job, ename, sal
   2 FROM        emp
   3 ORDER BY    job ;

Fri oct 24                                page 1

                        Job
                        Report

JOB          ENAME          SAL
-----
ANALYST      SCOTT          3000
              FORD          3000
CLERK        SMITH          800
              ADAMS         1100
...

                        Confidential
14 rows selected.
```

Pour désactiver l'affichage de TTITLE et BTITLE :

```
SQL> TTITLE OFF
SQL> BTITLE OFF
```

4.3.6 Exécuter un rapport formaté sous SQL*Plus

(cf. cours SQLP "Module 1 : Ordres SELECT Basiques" paragraphe 2.3.5 "Les commandes SQL*Plus manipulant les fichiers").

Les étapes de création d'un rapport :

- Ecrire un ordre SELECT dans l'invite de commande SQL.
Avant tout autre chose, il faut s'assurer que l'ordre s'exécute sans erreur et fournit le résultat souhaité.
L'ordre doit posséder une clause ORDER BY si des break sont utilisés.
- Sauvegarder l'ordre SELECT dans un fichier script à l'aide la commande SAVE.
- Charger le fichier script dans un éditeur à l'aide de la commande EDIT.
- Ajouter les commandes de formatages avant l'ordre SELECT. L'ordre SELECT ne doit pas contenir de commandes SQL*Plus.
- Vérifier la présence d'un caractère d'exécution (" ; " ou " / ") après l'ordre SELECT.
- Effacer le paramétrage du formatage après l'ordre SELECT.
- Sauvegarder le fichier script.
- Exécuter le contenu du fichier à l'aide de la commande START ou @.

Les abréviations des commandes SQL*Plus et les lignes blanches entre les commandes SQL*Plus sont acceptées dans le fichier script.

Le mot clé **REM** sert à marquer un commentaire dans le fichier script.

Exemple :

Créer un script qui crée un rapport affichant la fonction, le nom et le salaire des employés dont le salaire est inférieur à \$3000.

Ajouter l'entête « Employee Report » centré sur deux lignes et le pied de page « Confidential » centré.

Renommer le nom de la colonne JOB en « Job Category » placé sur deux lignes.

Renommer le nom de la colonne ENAME en « Employee ».

Renommer le nom de la colonne SAL en « Salary » et la formater comme suit : \$2,500.00.

```
SET PAGESIZE 37
SET LINESIZE 60
SET FEEDBACK OFF
TTITLE 'Employee|Report'
BTITLE 'Confidential'
BREAK ON job
COLUMN job      HEADING 'Job|Category' FORMAT A15
COLUMN ename    HEADING 'Employee' FORMAT A15
COLUMN sal      HEADING 'Salary' FORAMT $99,999.99

REM ordre select

SELECT          job, ename, sal
FROM            emp
WHERE           sal < 3000
ORDER BY        job, ename
/

REM effacer toutes les commandes de formatage

SET PAGESIZE 24
SET LINESIZE 80
SET FEEDBACK ON
TTITLE OFF
BTITLE OFF
CLEAR BREAK
COLUMN job      CLEAR
COLUMN ename    CLEAR
COLUMN sal      CLEAR
```

Exécution du script sous SQL*Plus :

```
SQL> START script

Fri Oct 24                                     page 1

                                Employee
                                Report

Job
Category                                Employee                                Salary
-----
CLERK                                ADAMS                                $1,100.00
                                JAMES                                $ 950.00
                                MILLER                                $1,300.00
                                SMITH                                $800.00
MANAGER                                BLAKE                                $2,850.00
                                CLARK                                $2,450.00
                                JONES                                $2,975.00
SALESMAN                                ALLEN                                $1,600.00
                                MARTIN                                $1,250.00
                                TURNER                                $1,500.00
```

	WARD	\$1,250.00
Confidential		

