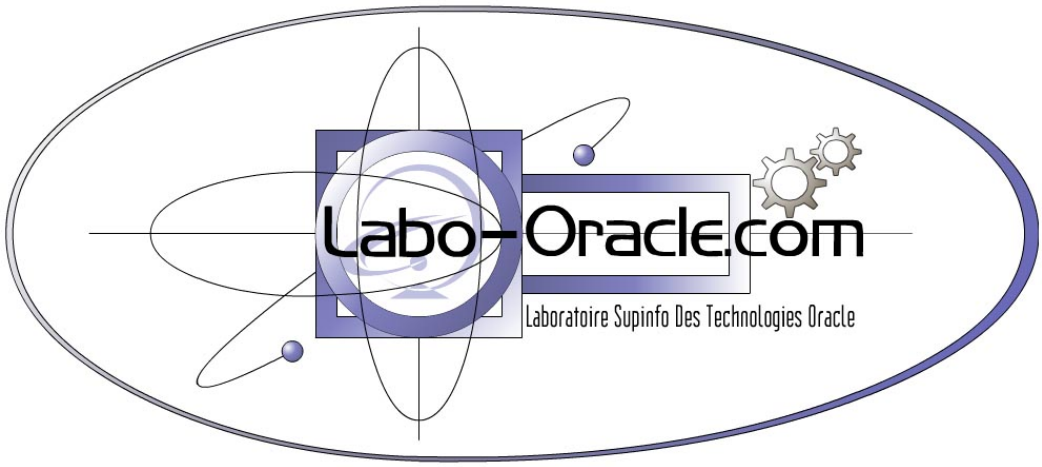


Module n° 2

TECHNIQUES DE RECUPERATION DES DONNEES

1Z0-007

Auteur : Andrei LANGEAC
Version 1.0 – 6 octobre 2004
Nombre de pages : 137

Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

Table des matières

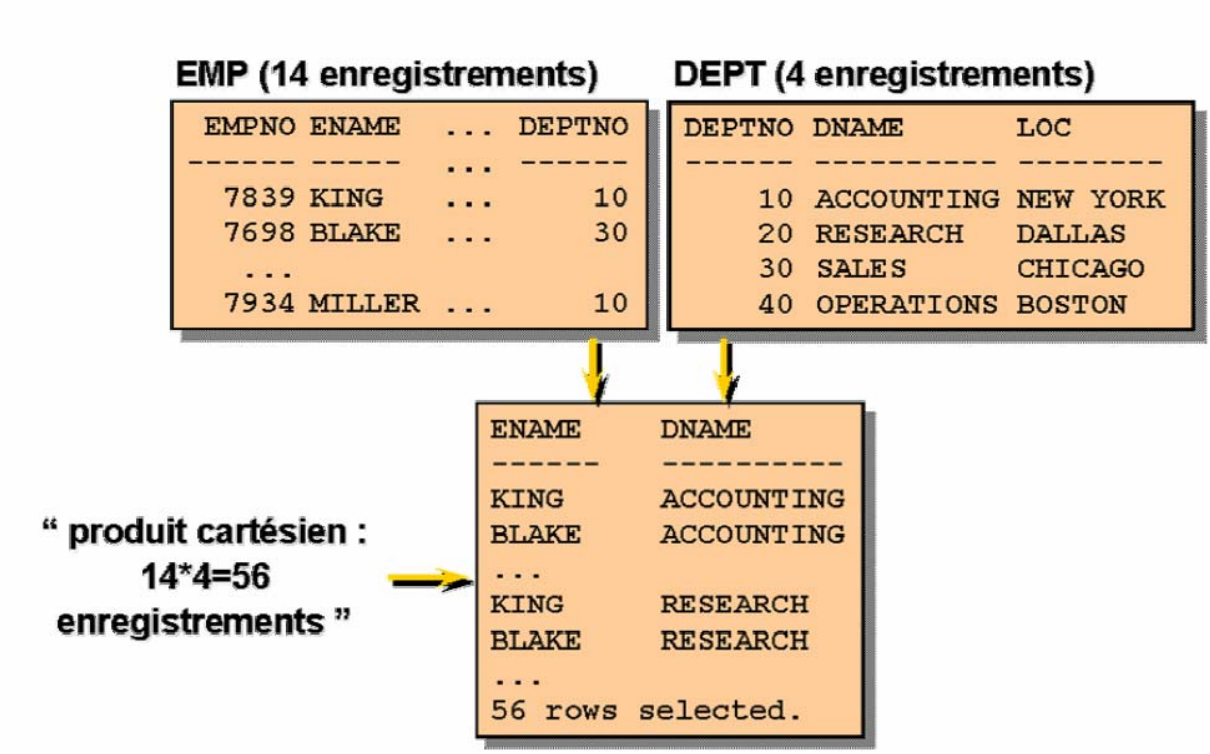
1. Affichage des données de plusieurs tables

1.1. Obtenir des données de plusieurs tables

1.1.1. 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.



1.1.2. 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 autre table. Cette relation est souvent établie entre des colonnes définies comme clé primaire et clé étrangère.

Il existe plusieurs types de jointures :

Les jointure propriétaires d'Oracle (8i et plus)	Les jointures complémentaires
Equijoin	Cross joins
Non-Equijoin	Natural joins
Outer join	Using clause
Self join	Full or two sided outer joins
	Arbitrary join conditions for outer joins

1.1.3. Relier les tables en utilisant la syntaxe d'Oracle.

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

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

Pour joindre les tables dans la clause **SELECT**

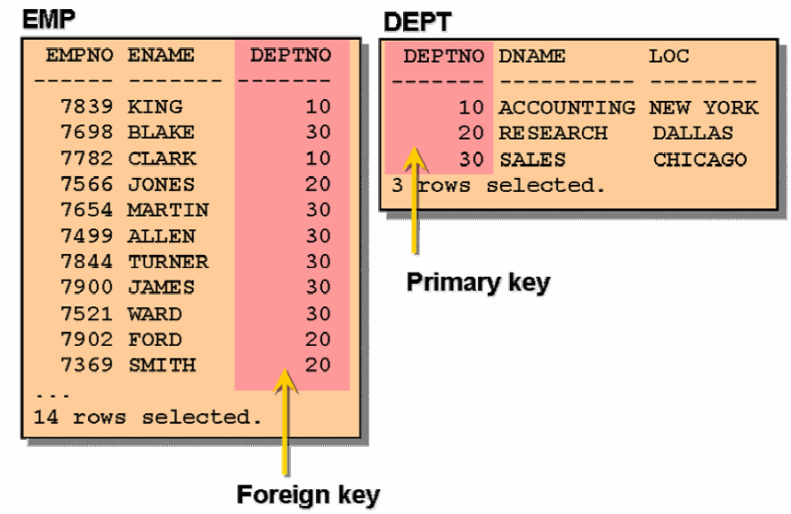
- On place les noms des tables avant les noms des colonnes, suivi d'un point (.)
- Vous devez précéder les noms de colonnes par les noms de tables si les mêmes colonnes apparaissent dans les différentes tables.
- Pour joindra *n* tables vous DEVEZ utiliser la condition de jointure *n-1*.

1.2. Equi-jointure

1.2.1. Récupération des résultats avec une equi-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.
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*;



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
---------- ---------- ---------- ---------- -------------
      7369 SMITH              20         20 DALLAS
      7499 ALLEN              30         30 CHICAGO
      7521 WARD               30         30 CHICAGO
      7566 JONES              20         20 DALLAS
      7654 MARTIN             30         30 CHICAGO
      7698 BLAKE              30         30 CHICAGO
…
14 ligne(s) sélectionnée(s).
```

Explication: 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. Ajout des condition de la recherche avec une clause AND

La condition peut être ajoutée 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]*

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'
SQL> /

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

Explication: Cette requête restreint les enregistrements à l'employé KING uniquement.

1.2.3. Utilisation les alias de tables

Pour alléger la requête, vous pouvez utiliser des alias pour les noms de table.
Les alias de tables

- peuvent posséder une longueur de plus de 30 caractères mais le mieux est d'avoir des alias d'un ou deux caractères
- Ces alias doivent être explicits.

- Ils sont valides juste pour la requête **SELECT** en cour.
- Les alias sont spécifiés dans la clause **FROM** après le nom de la table séparé par un espace.

Les alias de table améliorent 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).

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;

     EMPNO ENAME          DEPTNO     DEPTNO LOC
---------- ---------- ---------- ---------- -------------
      7369 SMITH              20         20 DALLAS
      7499 ALLEN              30         30 CHICAGO
      7521 WARD               30         30 CHICAGO
      7566 JONES              20         20 DALLAS
      7654 MARTIN             30         30 CHICAGO
      7698 BLAKE              30         30 CHICAGO
…
14 ligne(s) sélectionnée(s).
```

Exemple avec 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;

     EMPNO ENAME          DEPTNO     DEPTNO LOC
---------- ---------- ---------- ---------- -------------
      7369 SMITH              20         20 DALLAS
      7499 ALLEN              30         30 CHICAGO
      7521 WARD               30         30 CHICAGO
      7566 JONES              20         20 DALLAS
      7654 MARTIN             30         30 CHICAGO
      7698 BLAKE              30         30 CHICAGO
…
14 ligne(s) sélectionnée(s).
```

Explication : Les deux requêtes retournent le même résultat.

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* ;

```
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
```

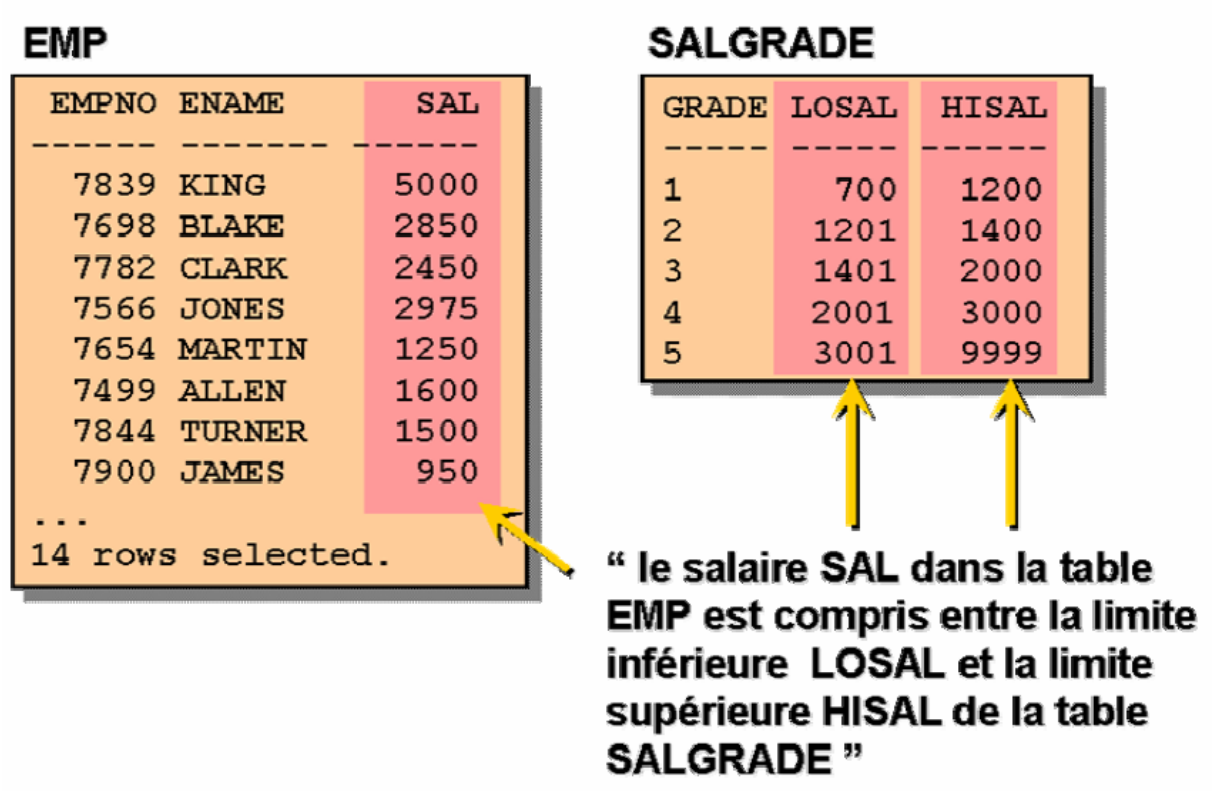
Explication : Cette requête affiche les ordres, les numéros d'item, le total de chaque item et le total de chaque ordre pour le client (customer) TKB SPORT SHOP.

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 client (customer) portant le nom TKB SPORT SHOP.

1.3. Autres types de jointures

1.3.1. Non equi-jointure

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



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
---------- ---------- ----------
SMITH              800          1
ADAMS             1100          1
JAMES              950          1
PAPIER            1000          1
WARD              1250          2
MARTIN            1250          2
MILLER            1300          2
…
14 ligne(s) sélectionnée(s).
```

Explication : 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 :
- Aucunes limites de 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. 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 (+) :

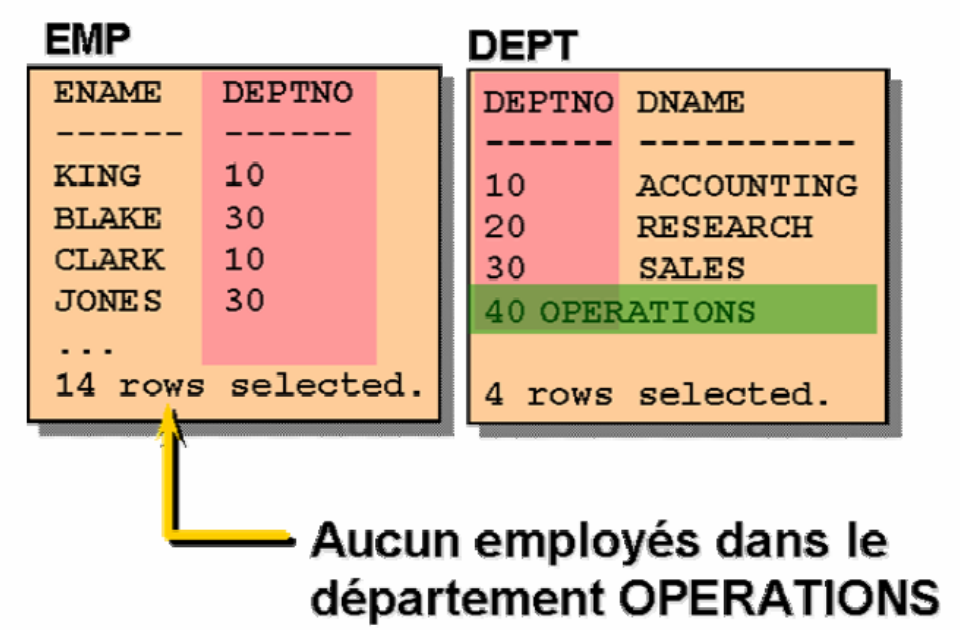
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1. column**(+)** = table2.column ;

Explication: Cette requête affiche tous les enregistrements de la colonne de *table1* même si ils ne respectent pas la condition de jointure.

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

Explication: Cette requête affiche tous les enregistrements de la colonne de *table2* 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 avec 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
---------- ---------- --------------
SMITH              20 RESEARCH
ALLEN              30 SALES
WARD               30 SALES
JONES              20 RESEARCH
MARTIN             30 SALES
BLAKE              30 SALES
…
14 ligne(s) sélectionnée(s).
```

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

Exemple avec une jointure externe :

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

ENAME          DEPTNO DNAME
---------- ---------- --------------
CLARK              10 ACCOUNTING
KING               10 ACCOUNTING
MILLER             10 ACCOUNTING
SMITH              20 RESEARCH
WARD               30 SALES
…
                   40 OPERATIONS
15 ligne(s) sélectionnée(s).
```

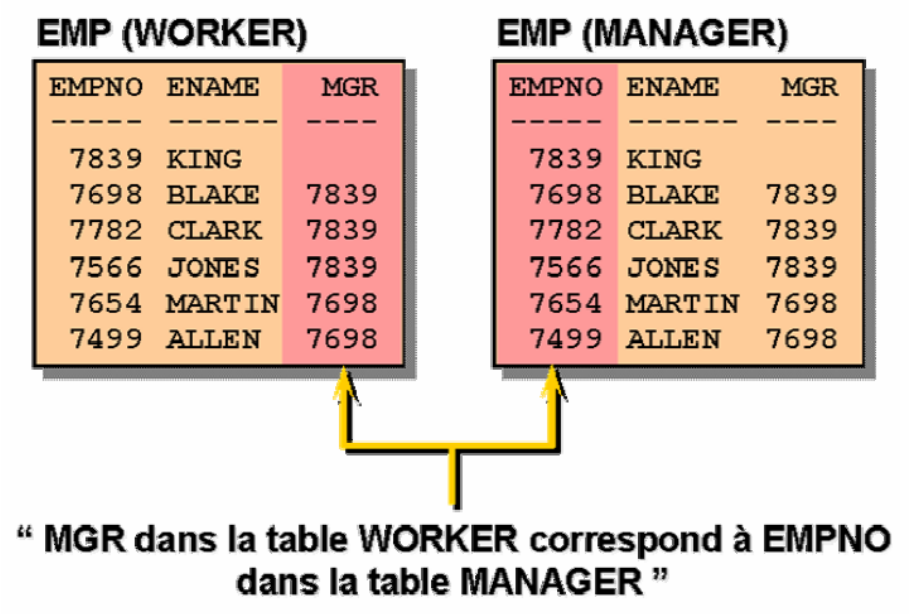
Explication: Cette requête affiche la liste des employés avec leur numéro et nom de département. Le département OPERATIONS apparaît cette fois dans le résultat malgré l'absence d'employé y travaillant.

1.3.3. 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 *alais1.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*).



Exemple :

```
SQL> SELECT  worker.ename||' travaille pour '||manager.ename REPORTS
  2  FROM    emp worker, emp manager
  3  WHERE   worker.mgr = manager.empno;

  REPORTS
  ----------------------------------
  SCOTT travaille pour JONES
  FORD travaille pour JONES
  ALLEN travaille pour BLAKE
  WARD travaille pour BLAKE
  JAMES travaille pour BLAKE
  TURNER travaille pour BLAKE
  …
  14 ligne(s) sélectionnée(s).
```

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

1.3.4. La jointure CROSS JOIN

Une condition de jointure croisée permet d'obtenir un produit en croix à partir de deux tables. C'est la même chose que le produit cartésien entre deux tables.

Exemple:

```
SQL>        SELECT       ename, dname
  2         FROM         emp, dept;

ENAME      DNAME
---------- --------------
SMITH      ACCOUNTING
ALLEN      ACCOUNTING
SMITH      RESEARCH
ALLEN      RESEARCH
SMITH      SALES
ALLEN      SALES
SMITH      OPERATIONS
ALLEN      OPERATIONS
…
136 ligne(s) sélectionnée(s).
```

Exemple:

```
SQL>        SELECT       ename, dname
  2         FROM         emp
  3         CROSS JOIN   dept;

ENAME      DNAME
---------- --------------
SMITH      ACCOUNTING
ALLEN      ACCOUNTING
SMITH      RESEARCH
ALLEN      RESEARCH
SMITH      SALES
ALLEN      SALES
SMITH      OPERATIONS
ALLEN      OPERATIONS
…
136 ligne(s) sélectionnée(s).
```

1.3.5. La jointure NATURAL JOIN

La jointure naturelle se base sur toutes les colonnes de deux tables qui possèdent le même nom et sélectionne les lignes qui possèdent les mêmes valeurs.

Exemple avec l'equi-jointure :

```
SQL>        SELECT              ename,dname
  2         FROM                emp, dept;

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
SMITH      RESEARCH
ADAMS      RESEARCH
…
14 ligne(s) sélectionnée(s).
```

Exemple avec une jointure **NATURAL JOIN** :

```
SQL>        SELECT              ename,dname
  2         FROM                emp
  3         NATURAL JOIN        dept;

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
SMITH      RESEARCH
ADAMS      RESEARCH
…
14 ligne(s) sélectionnée(s).
```

1.3.6. Création des jointures avec la clause USING

Si certaines colonnes possèdent les mêmes noms, mais les types de données ne correspondent pas, la clause **USING** est utilisée pour spécifier explicitement une colonne qui sera utilisé pour faire la jointure. Pour cette clause les alias et les noms de table ne doivent pas utilisés.

Exemple :

```
SQL>        SELECT        e.empno, e.ename, d.loc
  2         FROM          emp e JOIN dept d
  3         USING         (deptno);

     EMPNO ENAME      LOC
---------- ---------- -------------
      7369 SMITH      DALLAS
      7499 ALLEN      CHICAGO
      7521 WARD       CHICAGO
      7566 JONES      DALLAS
      7654 MARTIN     CHICAGO
     …
15 ligne(s) sélectionnée(s).
```

1.3.7. Création des jointures avec la clause ON

La nouveauté de la clause **ON** est qu'on spécifie explicitement la condition de la jointure.

Exemple :

```
SQL>        SELECT        e.empno, e.ename,d.dname, d.loc
  2         FROM          emp e JOIN dept d
  3         ON            (e.deptno=d.deptno)
SQL> /

     EMPNO ENAME      DNAME          LOC
---------- ---------- -------------- -------------
      7369 SMITH      RESEARCH       DALLAS
      7499 ALLEN      SALES          CHICAGO
      7521 WARD       SALES          CHICAGO
      7566 JONES      RESEARCH       DALLAS
      7654 MARTIN     SALES          CHICAGO
      7698 BLAKE      SALES          CHICAGO
      7782 CLARK      ACCOUNTING     NEW YORK
   …
15 ligne(s) sélectionnée(s).
```

1.3.8. Triple jointure avec la clause ON

La triple jointure vous permet de relier trois tables. L'utilisation de cette clause est identique à l'utilisation des clauses **WHERE** et **AND** lorsque vous reliez plusieurs tables.

Exemple :

```
SQL> SELECT       c.name,o.ordid,i.itemid,i.itemtot,o.total
  2  FROM         customer c
  3  JOIN         ord o
  4  ON           (c.custid=o.custid)
  5  JOIN         item t
  6  ON           (o.ordid=i.ordid);

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
```

1.3.9. Les jointures LEFT OUTER JOIN et RIGHT OUTER JOIN

La jointure **LEFT OUTER JOIN** est une alternative de la jointure externe avec le signe plus (+) placé à **droite** dans la clause **WHERE**.

La jointure **RIGHT OUTER JOIN** est une alternative de la jointure externe avec le signe plus (+) placé à **gauche** dans la clause **WHERE**.

Exemple :

```
SQL>        SELECT              e.ename, e.deptno, d.dname
            FROM                emp e
            RIGHT OUTER JOIN    dept d
            ON                  (e.deptno = d.deptno);

ENAME          DEPTNO DNAME
---------- ---------- --------------
SMITH              20 RESEARCH
ALLEN              30 SALES
WARD               30 SALES
JONES              20 RESEARCH
MARTIN             30 SALES
                   40 OPERATIONS
 …
14 ligne(s) sélectionnée(s).
```

1.3.10. La jointure FULL OUTER JOIN

SELECT *t1.colonne1, t2.colonne2*
FROM *table1 t1*
FULL OUTER JOIN *table2 t2*
ON *(t1.colonne3 = t2.colonne3) ;*

Cette jointure retrouve toutes les lignes de la *table1* même s'il n'y a pas de correspondance dans la *table2*, ainsi que toutes les lignes de la *table2* même s'il n'y a pas de correspondance dans la *table1*

Exemple :

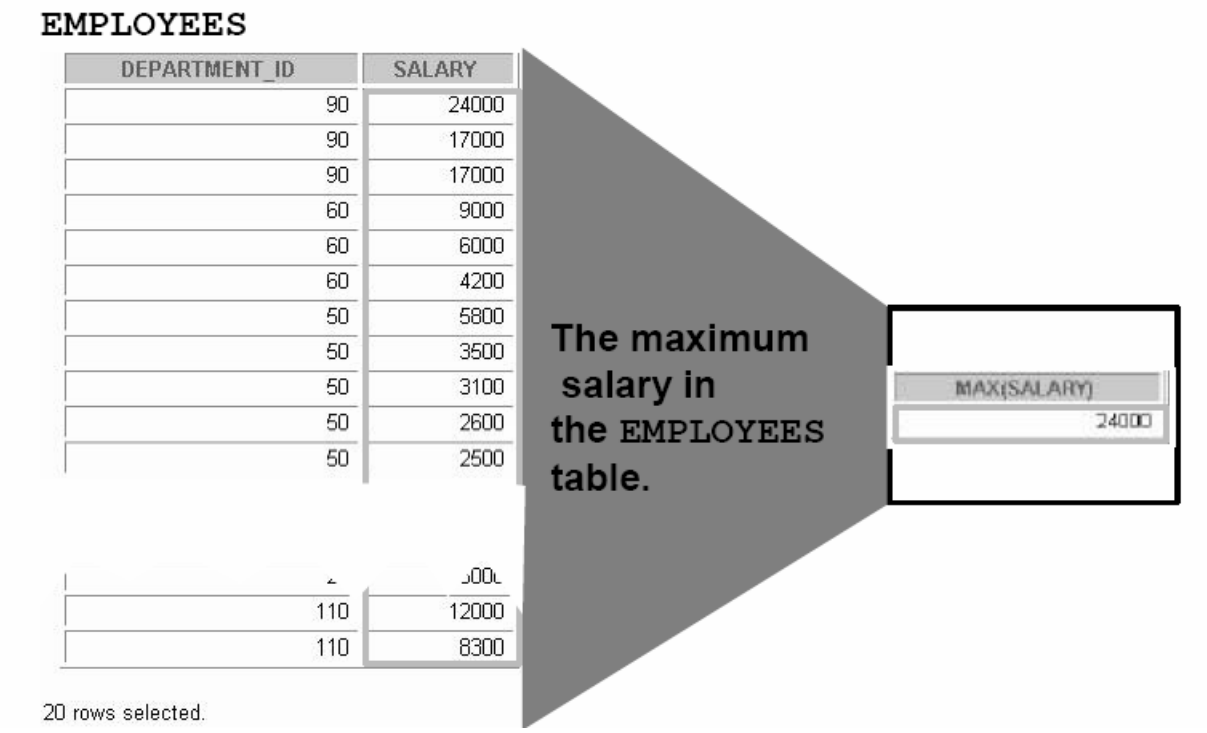
```
SQL> SELECT       e.ename,e.deptno,d.dname
  2  FROM         emp e
  3  FULL OUTER JOIN dept d
  4  ON           (e.deptno=d.deptno);


ENAME          DEPTNO DNAME
---------- ---------- --------------
MILLER             10 ACCOUNTING
KING               10 ACCOUNTING
CLARK              10 ACCOUNTING
...
PAPIER             30 SALES
JAMES              30 SALES
TURNER             30 SALES
...
WARD               30 SALES
ALLEN              30 SALES
THG
THG

                      store
                      STORE
                      OPERATIONS
                      STORE

22 row(s) selected.
```

2. LES FONCTIONS DE GROUPE

2.1. Fonctions de groupe

2.1.1. Qu'est ce qu'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. Différents types de fonctions de groupe

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

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

Le type de données des arguments peuvent 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**.

2.2.Utilisation des fonctions de groupe

2.2.1. Fonctions AVG et SUM

Exemple :

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

  AVG(SAL)   SUM(SAL)
---------- ----------
      1320       6600
```

Explication: Cette requête retourne la moyenne des salaries et la somme des salaries des employées dont la fonction commence par la chaîne de caractères.

2.2.2. Fonctions MIN et MAX

Exemple:

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

MIN(HIREDATE) MAX(HIREDATE)
--------      --------
17/12/80      12/01/83
```

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

2.2.3. Fonction COUNT

Cette fonction affiche le nombre de lignes dans la table spécifié dans la clause **SELECT**

Exemple :

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

  COUNT(*)
----------
         6
```

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

Exemple :

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

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

Explication : Cette requête retourne le nombre de départements (doublons inclus) de la table EMP.

2.2.4. Fonction DISTINCT

Cette fonction supprime les doublons.

Exemple :

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

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

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

2.2.5. NVL et fonctions de groupe

Exemple :

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

 AVG(COMM)
----------
      1470
```

Explication : Cette requête retourne la moyenne des commissions obtenues 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 :

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

AVG(NVL(COMM,0))
----------------
      864,705882
```

Explication : Cette requête retourne la moyenne des commissions obtenues 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 tous les employés en compte pour le calcul de la moyenne.

2.3. Création de groupes de données

2.3.1. Utilisation de 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 inclues 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**.

Exemple:

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

    DEPTNO   AVG(SAL)
---------- ----------
        10 2916,66667
        20       2175
        30 1485,71429
```

Explication: 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 **SELECT**.

Exemple :

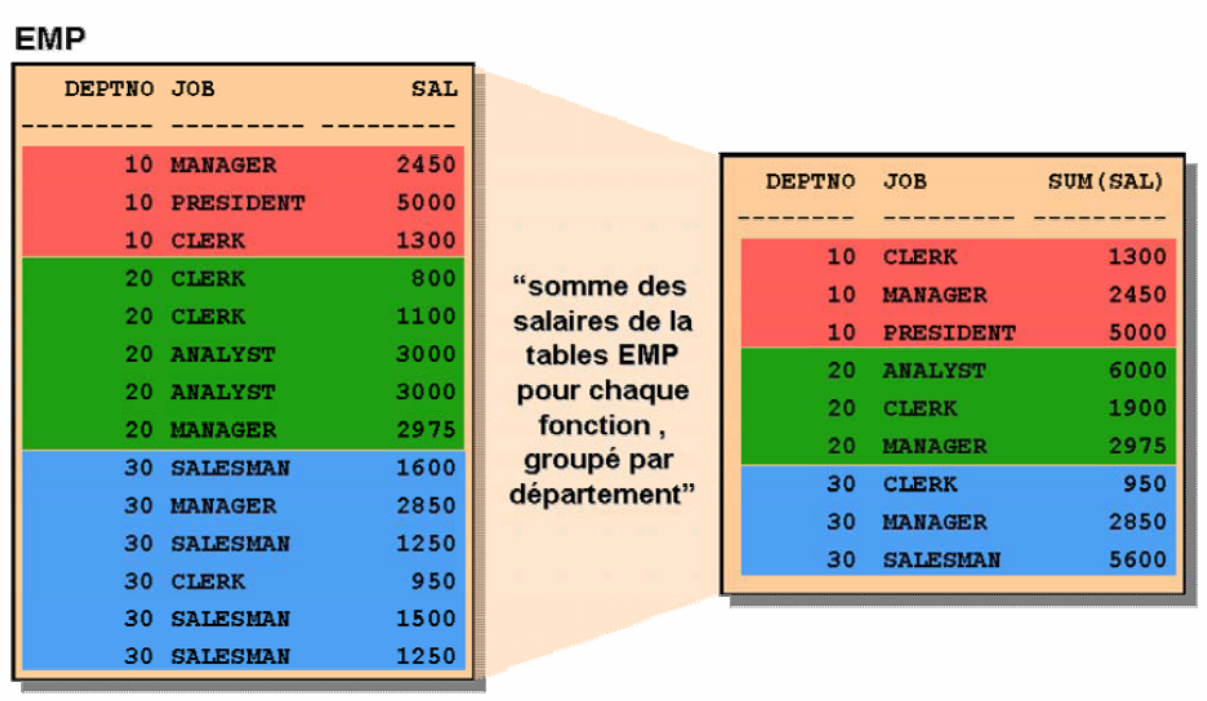
```
SQL>        SELECT       AVG(sal)
  2         FROM         emp
  3         GROUP BY     deptno ;

  AVG(SAL)
----------
2916,66667
      2175
1485,71429
```

2.3.2. 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* ;

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*.



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
        20 MANAGER         2975
        30 CLERK            950
  …
10 ligne(s) sélectionnée(s).
```

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

2.3.3. Utilisation de la clause HAVING

La clause **WHERE** n'acceptant pas les fonctions de groupe, la restriction du résultat des fonctions de groupe 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 :

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

    DEPTNO   MAX(SAL)
---------- ----------
        10       5000
        20       3000
```

Explication : Cette requête affiche les départements dont le salaire maximal est supérieur à $2900.

Exemple :

```
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
```

Explication: Cette requête affiche la somme des salaires des employés supérieure à $5000, et les jobs dont les cinq premières lettres sont différentes de la chaîne de caractères « SALES ». Le résultat est ordonné de façon décroissante sur les sommes des salaires.

2.3.4. 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))
-------------
   2916,66667
```

Explication : **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.

2.3.5. Les requêtes invalides.

Un alias de colonne ne peut pas être utilisé dans le **GROUP BY**. Sinon l'erreur suivante se produit : « ORA-00904: Nom de colonne non valide ».

Toutes les colonnes ou expressions dans la clause **SELECT**, qui ne sont pas l'argument 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)
       *
ERREUR à la ligne 1 :
ORA-00937: La fonction de groupe ne porte pas sur un groupe simple
```

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
      *
ERREUR à la ligne 3 :
ORA-00934: Fonction de groupe non autorisée ici
```

3. LES SOUS-REQUETES

3.1. Les sous requêtes basiques

3.1.1. Utilisation 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.

Pour combiner deux requêtes, il suffira de placer une requête à l'intérieur d'une autre. La requête à l'intérieure (ou 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:

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

ENAME
----------
SCOTT
KING
FORD
```

2975

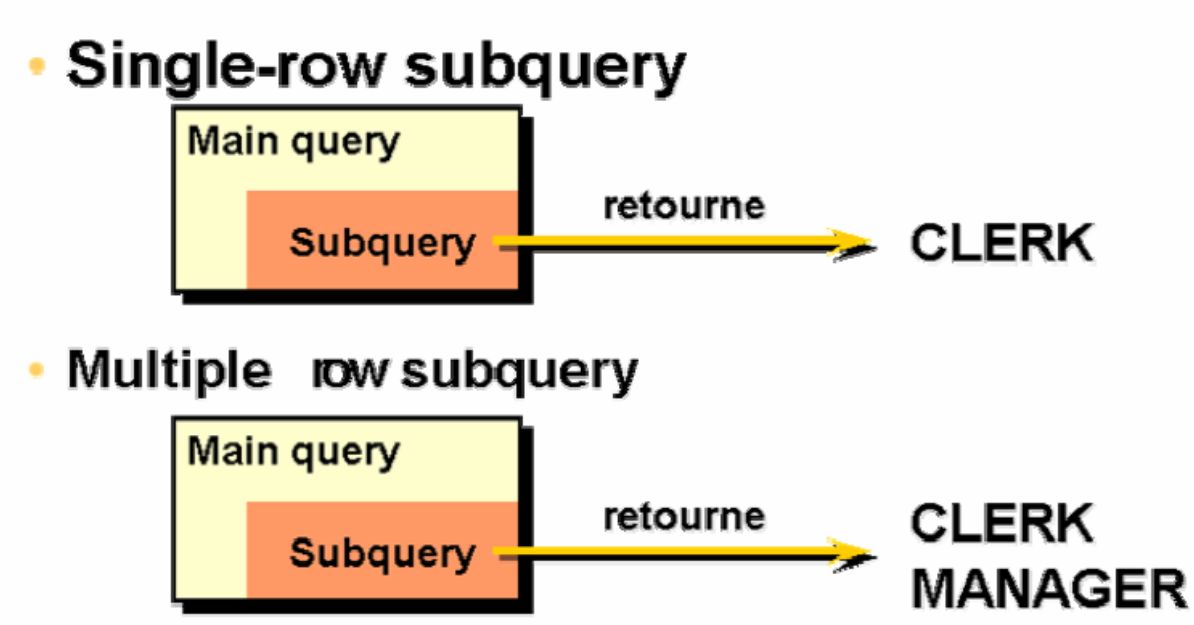
Explication: La sous-requête retourne le salaire de l'employé JONES. La requête principale compare le salaire des employés au salaire de l'employé JONES et ne retourne que ceux qui lui sont supérieurs.

Règles :
- 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. Types des sous-requêtes

Il existe deux types de sous-requête :

- Single-row : Retourne une valeur contenue dans une colonne.
- Multiple-row : Retourne plusieurs valeurs contenues dans une colonne.



3.2.Les sous-requêtes single-row

3.2.1. Exécution d'une sous-requête single-row

Une sous-requête single-row se situe dans la clause **WHERE** de la requête principale. Elle ne retourne qu'une seule ligne.
Une sous-requête single-row ne peut être utilisée qu'avec les opérateurs de comparaison suivants :
<, >, =, <=, >=, <>.

Exemple :

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

ENAME      JOB
---------- ---------
SMITH      CLERK
ADAMS      CLERK
JAMES      CLERK
MILLER     CLERK
```

Explication: 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.

Syntaxe :

SELECT *select_list*
FROM *table*
WHERE *expression single-row_comparison_operator* (
SELECT *select_list*
FROM *table*)
AND *expression single-row_comparison_operator* (
SELECT *select_list*
FROM *table*) ;

Exemple:

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


  6         AND sal > (                          1100
                                    SELECT sal
  7                                 FROM emp
  8                                 WHERE empno = 7876) ;

ENAME      JOB
---------- ---------
MILLER     CLERK
```

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

3.2.2. Utilisation des fonctions de groupes dans les sous-requêtes.

Des fonctions de groupes peuvent être utilisées dans une sous-requête pour opérer sur un groupe de valeurs.
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                  800
  3         WHERE        sal = (
                                    SELECT MIN(sal)
  4                                 FROM emp);

ENAME      JOB              SAL
---------- --------- ----------
SMITH      CLERK            800
```

Explication: 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. Utilisation de la clause HAVING dans les sous-requêtes.

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

Exemple :

```
SQL>        SELECT       deptno, MIN(sal)
  2         FROM         emp
  3         GROUP BY     deptno
  4         HAVING       MIN(sal) > (                  800

                                      SELECT MIN (sal)
  5                                   FROM emp
  6                                   WHERE deptno = 20);

    DEPTNO   MIN(SAL)
---------- ----------
        10       1300
        30        950
```

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

3.3.Les 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**.
Une sous-requête multiple-row ne peut pas contenir de 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 (                  800, 950, 1300

                                      SELECT MIN(sal)
  4                                   FROM emp
  5                                   GROUP BY deptno);

ENAME             SAL     DEPTNO
---------- ---------- ----------
SMITH             800         20
JAMES             950         30
MILLER           1300         10
```

Explication : 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);
```

3.3.1. Opérateur ANY

L'opérateur **ANY** compare une valeur à chaque valeur retournée par la sous-requête.

Exemple :

```
SQL>        SELECT        empno, ename, job
  2         FROM          emp
  3         WHERE         sal < ANY (                1300, 800, 950

                                      SELECT sal
  4                                   FROM emp
  5                                   WHERE job = 'CLERK')
  6         AND           job <> 'CLERK';

     EMPNO ENAME      JOB
---------- ---------- ---------
      7521 WARD       SALESMAN
      7654 MARTIN     SALESMAN
      7952 PAPIER     SALESMAN
```

Explication : Cette requête affiche les employés dont la fonction n'est pas CLERK et dont le salaire est plus petit que celui des 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.

Opérateur	Signification
< **ANY**	Signifie plus petit que le minimum.
> **ANY**	Signifie plus grand que le maximum.
= **ANY**	Est équivalent à **IN**.

3.3.2. Opérateur ALL

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

Exemple :

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

     EMPNO ENAME      JOB
---------- ---------- ---------
      7499 ALLEN      SALESMAN
      7566 JONES      MANAGER
      7698 BLAKE      MANAGER
      7782 CLARK      MANAGER
      7788 SCOTT      ANALYST
      7839 KING       PRESIDENT
      7844 TURNER     SALESMAN
      7902 FORD       ANALYST

8 ligne(s) sélectionnée(s).
```

Explication : 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.

Opérateur	Signification
> **ALL**	Signifie plus grand que le maximum.
< **ALL**	Signifie plus petit que le minimum

3.3.3. Les valeurs nulles

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**.

3.3.4. L'utilisation de sous-requêtes dans la clause FROM.

Des sous-requêtes 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       e.ename, e.sal, e.deptno, b.salavg
  2         FROM         emp e, (
                                    SELECT deptno, AVG(sal) salavg
  3                                 FROM emp
  4                                 GROUP BY deptno) b
  5         WHERE e.deptno = b.deptno
  6         AND e.sal < b.salavg;

ENAME              SAL     DEPTNO     SALAVG
---------- ---------- ---------- ----------
CLARK            2450         10 2916,66667
MILLER           1300         10 2916,66667
SMITH             800         20       2175
ADAMS            1100         20       2175
PAPIER           1000         30 1485,71429
MARTIN           1250         30 1485,71429
JAMES             950         30 1485,71429
WARD             1250         30 1485,71429

8 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche, pour chaque employé ayant 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. CREATION DES RAPPORTS AVEC iSQL*PLUS.

4.1. Les variables de substitution

Lorsqu'une requête est exécutée un certain nombre de fois avec des valeurs différentes à chaque fois, la requête doit être modifiée et lancée autant de fois qu'il y a de valeurs différentes.
Les variables de substitutions serviront à saisir les valeurs de l'utilisateur à chaque lancement de la requête au lieu de modifier manuellement les valeurs.
Les variables de substitution sont des contenants dans lesquels sont stockées temporairement des valeurs.
L'utilisation des variables de substitution en SQL*Plus consiste à demander à l'utilisateur d'entrer une valeur qui sera substituée à la variable correspondante.
Une variable de substitution est une variable définie et nommée par le programmeur (celui qui écrit la requête).
Le nom de variable est précédé d'un à deux '**&**'.
Une variable de substitution peut être placée n'importe où dans un ordre SQL exceptée en tant que premier mot de l'ordre. Une variable de substitution ne peut pas remplacer une clause **SELECT**.

4.1.1. Utilisation d'ampersand &

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

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

Exemple :

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

Entrez une valeur pour departement : 10

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

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

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

4.1.2. Substitution de chaînes de caractères et de dates.

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

Exemple :

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

Entrez une valeur pour job_title : ANALYST

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

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

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

On peut utiliser les fonctions **UPPER** et **LOWER** sur des variables de substitution.

UPPER('&*user_variables*')
LOWER('&*user_variables*')

L'utilisation de ces fonctions permet de ne pas tenir compte de la casse de la valeur saisie par l'utilisateur.
Si la variable de substitution attend la saisie d'une date, elle doit être saisie au format par défaut DD-MON-YY.

4.1.3. Utilisation de double ampersand &&

Si une variable est précédée de deux caractères '**&**', alors SQL*Plus ne demandera la saisie de la valeur qu'une seule fois lors de l'exécution d'une requête.

&&user-variable → pour les valeurs de type numérique
'&&user-variable**'** → pour les valeurs de type date et chaînes de caractères

On utilise le double '&', lorsqu'une variable est utilisé plusieurs fois dans une requête.

Exemple :

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

Entrez une valeur pour column_name : deptno

ancien   1 : SELECT empno, ename, job, &&column_name
nouveau   1 : SELECT empno, ename, job, deptno

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

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

Explication : L'utilisateur est appelé qu'une seule fois à saisir la variable *column_name* qui apparaît deux fois dans l'ordre SQL. La valeur saisie par l'utilisateur (*deptno*) est utilisée pour les deux apparitions de la variable dans le code.

La valeur est stockée dans la variable jusqu'à la fin de la session ou jusqu'à ce qu'elle soit indéfinie.

4.2. Définition des variables de substitution

4.2.1. Utilisation de la commande DEFINE

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

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

Exemple :

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

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

Exemple :

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

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

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

Explication : Cette requête affiche le département dont le nom est défini par la variable *deptname et* dont la valeur est « *sales* ». La variable *deptname* définie par la commande **DEFINE** s'utilise comme n'importe quelle variable.

4.2.2. Utilisation de la commande UNDEFINE

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 soient recréées au démarrage.

Exemple :

```
SQL>       UNDEFINE deptname
SQL>       DEFINE deptname
SP2-0135: le symbole deptname est INDEFINI
```

Explication : 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.

4.2.3. Utilisation de la commande VERIFY

La commande **VERIFY** permet d'afficher les variables de substitution avant et après le remplacement effectué par iSQL*PLUS.

SET VERIFY ON

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

4.3. Personnalisation de l'environnement iSQL*PLUS

4.3.1. Utilisation de la commande SET

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

Fonction	Description
ARRAY[**SIZE**] { 20 | *n* }	Définit la taille « database data fetch »
FEED[**BACK**] { 6 | *n* | **OFF** | **ON** }	Nombre d'enregistrements retournés par la requête à partir duquel SQL*Plus affiche le message des enregistrements sélectionnés
HEA[**DING**] { **OFF** | **ON** }	Afficher les entêtes de colonnes
LONG { 80 | *n* }	Longueur maximale lors de l'affichage d'un **LONG**

4.3.2. Les commandes de formatage

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

Fonction	Description
COL[**UMN**] [*column option*]	Contrôle le format des colonnes
BRE[**AK**] [**ON** *report-element*]	Supprime les doublons et divise les enregistrements en section
TTI[**TLE**] [*text* | **OFF** | **ON**]	Spécifie l'entête de chaque page de l'état
BTI[**TLE**] [*text* | **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 réfèreront 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. Utilisation de la commande COLUMN

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

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

Les options de la commande **COLUMN** :

Fonction	Description
CLE[**AR**] *column_name*	Efface tous les formats de la colonne *column_name*.
FOR[**MAT**] *format*	Formate une colonne avec les models de formatage de la commande
HEA[**DING**] *text*	Affecte 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**] { *align* }	Justifie l'entête d'une colonne : **R**[**IGHT**], **L**[**EFT**] ou **C**[**ENTER**]
NOPRI[**NT**] | **PRI**[**NT**]	Affiche ou pas une colonne
NUL[**L**] { *text* }	Affiche la chaîne de caractères *text* à la place des valeurs nulles
TRU[**NCATED**]	Tronque une chaîne de type **CHAR**, **VARCHAR2**, **LONG** ou **DATE** qui est trop grande pour une colonne.
WRA[**PPED**]	Met 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 :

Fonction	Description
COL[**UMN**] *column_name*	Affiche le paramétrage de la colonne c*olumn_name*
COL[**UMN**]	Affiche le paramétrage de toutes les colonnes
COL[**UMN**] column_name **CLE**[**AR**]	Efface le paramétrage de la colonne *column_name*
CLE[**AR**] **COL**[**UMN**]	Efface 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.

Le tableau ci-dessous représente les différents formats de colonnes :

Elément	Description	Exemple	Résultat
9	Enlève les zéros non nécessaires au début	999999	1234
0	Force à mettre les zéros au début	099999	001234
$	Signe dollar flottant	$999999	$1234
L	Devise locale	L999999	FF1234
.	Position du point des décimales	999999.99	1234.00
,	Séparateur des milliers	999,999	1,234

4.3.4. Utilisation de 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*]]

Exemple :

```
SQL>        BREAK ON
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

…
14 ligne(s) sélectionnée(s).
```

Explication : 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. Utilisation des commandes TTITLE et BTITLE

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

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

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

BTIL[**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.

Exemple :

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

Lu Aou 09                                                  page    1

                                     Job
                                    Report

JOB       ENAME             SAL
--------- ---------- ----------
ANALYST   SCOTT            3000
          FORD             3000
CLERK     SMITH             800
          ADAMS            1100
          MILLER           1300
          JAMES             950


                                  Confidential
…

14 ligne(s) sélectionnée(s).
```

Pour désactiver l'affichage de **TTITLE** et **BTITLE** :

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

4.3.6. Exécution des rapports formatés

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 formatage 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 :
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 FEEDBACK OFF
TTITLE 'Employee|Report'
BTITLE 'Confidential'
BREAK ON job
COLUMN job HEADING 'Job|Category'
COLUMN ename HEADING 'Employee'
COLUMN sal Heading 'Salary' FORMAT $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 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

Lu Aou 09                                              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
                                  Confidential
```