SQL
SELECT

TABLE DES MATIERES

1 LES BASES DU SQL

1.1 Le SELECT

L'ordre **SELECT** sert à extraire des données de la base de données.

```
SELECT colonne
FROM table;
```

Un ordre SELECT est composé de deux clauses :

- La clause **SELECT** qui spécifie les colonnes à sélectionner,
- La clause **FROM** qui spécifie la table où sont situées les colonnes sélectionnées.

L'ordre SELECT permet de sélectionner une ou plusieurs lignes, une ou plusieurs colonnes, et des colonnes dans des tables différentes, créant une relation entre les données des colonnes.

On exécute une requête SQL en la terminant par les caractères " **;** ".

La bonne pratique veut que l'on mette chaque clause sur une ligne différente, on note les mots réservés en majuscule et les autres mots en minuscule.

Remarque : En SQL la casse n'est pas prise en compte.

Le caractère " * " placé dans la clause SELECT sert à récupérer toutes les colonnes de toutes les tables de la clause FROM.

L'ordre des colonnes spécifiées dans la clause SELECT correspond à l'ordre dans lequel les résultats seront récupérés.

```
SELECT *
FROM dep;

DNUM        DNOM             DLOC
------      --------------   ---------
10          ACCOUNTING       NEW YORK
20          RESEARCH         DALLAS
30          SALES            CHICAGO
40          OPERATIONS       BOSTON
```

Toutes les colonnes de la table DEP sont sélectionnées et affichées.

```
SELECT dnum, dloc
FROM dep;

DNUM       DLOC
------     ------------
10         NEW YORK
20         DALLAS
30         CHICAGO
40         BOSTON
```

Les colonnes DNUM et DLOC de la table DEP sont sélectionnées et affichées.

1.2 Les opérateurs arithmétiques

Les opérateurs arithmétiques peuvent être utilisés pour effectuer des opérations sur les dates ou les nombres.
Ils peuvent être utilisés dans toutes les clauses sauf la clause FROM.
Les opérateurs arithmétiques disponibles sont les suivants :

Opérateur	Description
+	Additionner
-	Soustraire
*	Multiplier
/	Diviser

Les règles mathématiques s'appliquent normalement :
Les opérateurs arithmétiques * et / sont prioritaires par rapport aux opérateurs arithmétiques + , – .
Des parenthèses peuvent être utilisées pour clarifier les calculs et modifier l'ordre d'évaluation.

```
SELECT nom, sal, 12*sal+100
FROM emp;

NOM        SAL   12*SAL+100
----       ----  ----------
KING       5000  60100
BLAKE      2850  34300
...
```

```
SELECT nom, sal, 12*(sal+100)
FROM emp;

NOM        SAL    12*(SAL+100)
----       ----   ------------
KING       5000   61200
BLAKE      2850   35400
...
```

1.3 Les valeurs nulles

Une valeur nulle est une valeur non assignée, inconnue ou inapplicable.
Elle n'est ni équivalente à zéro ni à un espace !

```
SELECT nom, job, sal, comm
FROM emp;

NOM        JOB        SAL   COMM
-----      ----       ----  ----
KING       PRESIDENT  5000
BLAKE      MANAGER    2850
TURNER     SALESMAN   1500  0
…
```

La valeur de la commission de KING et de BLAKE est nulle, tandis que la commission de TURNER est de 0.

Si une expression arithmétique contient une valeur nulle, alors le résultat de l'expression sera nul.

1.4 Personnalisation des requêtes

1.4.1 Alias

Un **alias** de colonne est une chaîne de caractère qui se substitue au nom de la colonne pour le traitement et l'affichage de la colonne.
Le mot-clé (optionnel) **AS** permet d'assigner un alias à une colonne.

```
SELECT colonne1 AS alias1, colonne2 AS alias2
FROM table;
```

L'utilisation des alias de colonne clarifie et allège les requêtes SQL.
Il est fortement conseillé de les utiliser.
Les alias servent également à "renommer" les colonnes lors de l'affichage.

1.4.2 Concaténation

La fonction CONCAT(partie1, partie2) sert à concaténer des colonnes ou des chaînes de caractères à d'autres colonnes.

```
SELECT CONCAT(nom,dnum) AS "Password"
FROM emp;

Password
--------
SMITH20
ALLEN30
```

1.4.3 Chaine de caractères littérale

Des chaînes de caractères peuvent être ajoutées dans une clause SELECT.
Ces chaînes sont constantes et s'affichent pour chaque ligne du résultat. Elles peuvent être de type caractère, nombre ou date. Les chaînes de type caractères et date doivent être inclues entre simple côtes.
Une chaîne de caractère placée dans un ordre SELECT doit être mise entre simples côtes.

```
SELECT nom
     || ' travaille au département '
     || dnum AS "Localisation"
FROM emp;

Localisation
-----------------------------------
SMITH travaille au département 20
ALLEN travaille au département 30
WARD travaille au département 30
JONES travaille au département 20
```

Grâce à l'opérateur de concaténation, toutes les colonnes (y compris la chaîne de caractères) ne forment qu'une seule colonne

1.4.4 Elimination des doublons

Le mot-clé **DISCTINCT** élimine les doublons dans le résultat de la requête.
Un **doublon** est un enregistrement qui se répète plusieurs fois.
Le mot-clé DISTINCT est utilisé dans la clause SELECT.

```
SELECT dnum
FROM emp;

DNUM
----------
10
20
20
```

Certains départements apparaissent plusieurs fois dans le résultat. La requête affiche tous les doublons.
Grâce au mot-clé DISTINCT, les départements n'apparaissent qu'une seuls fois dans le résultat.

```
SELECT DISTINCT dnum
FROM emp;

DEPTNUM
----------
10
30
20
```

2 RESTREINDRE LES ENREGISTREMENTS

2.1 Le WHERE

La clause WHERE restreint la requête aux enregistrements qui respectent sa ou ses conditions.
La clause WHERE correspond à une sélection : elle restreint les enregistrements (ou lignes).

```
SELECT nom, job, dnum
FROM emp
WHERE job = 'CLERK';

NOM        JOB        DNUM
-----      ------     -----
SMITH      CLERK      20
ADAMS      CLERK      20
JAMES      CLERK      30
MILLER     CLERK      10
```

Cette requête affiche le nom, le job et le numéro de département des employés dont la fonction est "CLERK".
Attention :
La comparaison de deux chaînes de caractères est sensible à la casse (‘CLERK’ est différent de ‘Clerk’ ou encore ‘clerk’, cf: les fonctions opérant sur les caractères)
Les dates sont sensibles au format. ('01/01/81' est différent de '01/01/1981').
Les chaînes de caractères et les dates doivent être placées entre simples côtes.

2.2 Opérateurs de comparaison

2.2.1 Expressions de comparaison

Opérateur	Signification
=	Egal à
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
<>	Différent de

Ces opérateurs ne tiennent pas compte des valeurs nulles (NULL VALUE).

```
SELECT nom, sal
FROM emp
WHERE sal > 2000;

NOM         SAL
-----       -----
JONES       2975
BLAKE       2850
CLARK       2450
SCOTT       3000
```

```
SELECT dnum, loc
FROM dept
WHERE loc <> 'NEW YORK';

DNUM        LOC
-----       -----
20          DALLAS
30          CHICAGO
40          BOSTON
```

Cette requête affiche les départements qui ne sont pas localisés à NEW YORK.

2.2.2 L'opérateur BETWEEN

L'opérateur **BETWEEN..AND** permet d'afficher des enregistrements basés sur une tranche de valeurs

Les limites peuvent être des nombres, des caractères, des dates. Dans le cas où il s'agirait de caractères ou de dates, les limites doivent être placées entre simples côtes.
L'opérateur BETWEEN est inclusif (ces limites sont incluses dans la tranche de valeurs possibles).
Le serveur traduit l'opérateur BETWEEN..AND comme la combinaison de deux conditions reliées par l'opérateur AND : (a >= lower_limit) AND (a <= higher_limit)
L'utilisation de l'opérateur BETWEEN..AND n'apporte aucun bénéfices de performance.

```
SELECT nom, sal
FROM emp
WHERE sal BETWEEN 1000 AND 1500;

NOM         SAL
----        ---
WARD        1250
MARTIN      1250
TURNER      1500
```

```
SELECT nom
FROM emp
WHERE nom BETWEEN 'S' AND 'W';

NOM
-------
SMITH
SCOTT
TURNER
```

Cette requête affiche les employés dont le nom est situé dans la tranche "S" à "W".
Remarque : L'employé WARD n'apparaît pas car "WA" est plus grand que "W".

2.2.3 L'opérateur IN

L'opérateur IN permet d'afficher des enregistrements appartenant à une liste de valeurs.
Les valeurs de la liste peuvent être des nombres, des caractères, des dates. Dans le cas où il s'agirait de caractères ou de dates, les valeurs doivent être placées entre simples côtes.

```
SELECT empno, nom, mgr
FROM emp
WHERE mgr IN (7902, 7566, 7788);

EMPNO  NOM        MGR
-----  ----       -----
7369   SMITH      7902
7788   SCOTT      7566
7876   ADAMS      7788
7902   FORD       7566
```

Cette requête affiche les employés dont le numéro de manager est 7902 ou 7566 ou 7788.

2.2.4 L'opérateur LIKE

L'opérateur LIKE permet de faire des recherches de caractères spécifiques dans une chaîne de caractères données.
Le symbole ‘_’ représente un seul caractère quelconque.
Le symbole ‘%’ représente une série de zéros ou de caractères.
L'opérateur LIKE peut être utilisé comme un raccourci de plusieurs conditions BETWEEN..AND

```
SELECT nom
FROM emp
WHERE nom LIKE '__A%';

NOM
----------
BLAKE
CLARK
ADAMS
```

Cette requête affiche les employés dont le troisième caractère de leur nom est un ‘A’ (majuscule !) suivie d'une chaîne de n caractères.

2.2.5 L'opérateur IS NULL

L'opérateur IS NULL permet d'afficher les enregistrements dont certains champs contiennent des valeurs nulles. Une valeur nulle signifie que la valeur n'est pas disponible, non assignée, inconnue ou inapplicable. Encore une fois, NULL n'est ni égal à 0, ni à aucune chaine de caractères, même vide.

```
SELECT nom, mgr
FROM emp
WHERE mgr IS NULL;

NOM    MGR
----   ---
KING
```

Cette requête affiche les employés qui ne possèdent pas de manager.

2.3 Les opérateurs logiques

2.3.1 L'opérateur AND

L'opérateur AND permet d'afficher les enregistrements qui vérifient toutes les conditions impliquées dans l'expression.

Résultats d'une combinaison de deux conditions AND : **AND**	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Résultats d'une combinaison de deux conditions AND : **AND**	TRUE	FALSE	NULL	FALSE	FALSE	FALS
TRUE	TRUE	FALSE	NULL	NULL	NULL	FALS

```
SELECT nom, job, sal
FROM emp
WHERE sal >= 1100
AND job = 'CLERK';

NOM         JOB         SAL
----        ---         ---
ADAMS       CLERK       1100
MILLER      CLERK       1300
```

Cette requête affiche les employés dont la fonction est CLERK et dont le salaire est supérieur ou égal à $1100.

Résultats d'une combinaison de deux conditions AND :

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

2.3.2 L'opérateur OR

L'opérateur OR permet d'afficher les enregistrements qui vérifient au moins une des conditions impliquées dans l'expression.

```
SELECT nom, job, sal
FROM emp
WHERE sal >= 1100 OR job = 'CLERK';

NOM         JOB         SAL
---------- ---------- ----
SMITH       CLERK       800
ALLEN       SALESMAN    1600
```

Cette requête affiche les employés dont la fonction est CLERK ou dont le salaire est supérieur ou égal à $1100.

2.3.3 L'opérateur NOT

L'opérateur NOT permet d'afficher les enregistrements qui ne vérifient pas la condition impliquée dans l'expression.
L'opérateur NOT retourne TRUE si la condition de l'expression logique est FALSE.
L'opérateur NOT ne peut être utilisé qu'avec les opérateurs de comparaison IN, LIKE, BETWEEN et IS NULL.

Résultats d'une combinaison de deux conditions NOT :

NOT	TRUE	FALSE	NULL
TRUE	FALSE	TRUE	NULL

```
SELECT nom, comm
FROM emp
WHERE comm IS NOT NULL;

NOM        COMM
---------- ----
ALLEN      300
WARD       500
MARTIN     1400
TURNER     0
```

Cette requête affiche la liste des employés touchant une commission.

2.3.4 Ordres d'évaluation des opérateurs

Ordre d'évaluation des opérateurs :
Evaluer en premier : opérateurs de comparaison
Evaluer en second : opérateurs logiques (dans cet ordre NOT, AND, OR)
Les conditions entre parenthèses sont évaluées en premier.

```
SELECT nom, job, sal
FROM emp
WHERE (job='SALESMAN'
      OR job='PRESIDENT')
AND sal>1500;

NOM        JOB         SAL
---------- --------- -----
ALLEN      SALESMAN   1600
KING       PRESIDENT  5000
```

2.4 La Clause ORDER BY

La clause **ORDER BY** permet d'afficher les enregistrements sélectionnés dans l'ordre croissant ou décroissant.

L'ordre par défaut est croissant.

```
SELECT nom, job, deptno, hiredate
FROM emp
ORDER BY hiredate;

NOM        JOB   DEPTNO     HIREDATE
---------  ---   ------     --------
SMITH      CLERK    20      17/12/80
ALLEN      SALESMAN 30      20/02/81
WARD       SALESMAN 30      22/02/81
JONES      MANAGER  20      02/04/81
```

Les enregistrements peuvent être triés sur une colonne qui n'a pas été sélectionnée dans la clause SELECT.

```
SELECT column1, column2
FROM table
ORDER BY column3;
```

Le mot-clé **DESC** permet d'afficher les enregistrements sélectionnés dans l'ordre décroissant.
Les valeurs nulles sont affichées en premier lors d'un tri décroissant.

Les enregistrements peuvent être triés sur un alias de colonne ou une expression.

```
SELECT column1 AS alias1, column2 AS alias2
FROM table
ORDER BY alias1;
```

Les enregistrements peuvent être triés sur plusieurs colonnes.

```
SELECT nom, deptno, sal
FROM emp
ORDER BY deptno, sal DESC;

NOM        DEPTNO     SAL
---------- ---------- ----
KING       10         5000
CLARK      10         2450
MILLER     10         1300
SCOTT      20         3000
```

```
FORD        20          3000
```

Les enregistrements sont triés dans l'ordre croissant par rapport au numéro de département et, à numéro de département identique, dans l'ordre décroissant par rapport au salaire.

3 LES FONCTIONS

3.1 Les types de fonctions SQL

Une fonction SQL effectue une opération sur des données.
Les fonctions SQL peuvent être utilisées pour formater des dates et des nombres pour l'affichage, et pour convertir des types de données de colonne.
Une fonction SQL peut accepter un à plusieurs arguments mais ne retourne toujours qu'une seule valeur.

Il existe deux types de fonction SQL :

Single-row functions :

Elles opèrent sur des enregistrements seuls et produisent un résultat par enregistrement. Exemples : LOWER(), LENGTH(), MOD()

Multiple-row functions :

Elles opèrent sur un groupe d'enregistrements et produisent un résultat par groupe d'enregistrements. Elles sont aussi appelées **fonctions de groupe**. Exemple : SUM()

3.2 Les fonctions SQL single-row

Un argument d'une fonction single-row peut-être : une constante, une variable défini par l'utilisateur, une colonne ou une expression.
Elles peuvent :

retourner un type différent de son ou ses argument(s),
être utilisées dans les clauses SELECT, WHERE et ORDER BY,
s'imbriquer.

Les fonctions single-row sont classées en cinq types :

les fonctions opérant sur les caractères
les fonctions opérant sur les nombres
les fonctions opérant sur les dates
les fonctions de conversion de types de données
les fonctions générales

3.3 Les fonctions opérant sur les caractères

Les fonctions opérant sur des caractères sont divisées en deux sous-groupes :

Les fonctions de conversion de casse
Les fonctions de manipulation des caractères

3.4 Les fonctions de conversion de casse

Les fonctions de conversion de la casse permettent d'afficher ou d'utiliser les données dans une casse différente de celle qu'elles possèdent dans la table.

LOWER(column | expr)	Convertit une chaîne de caractères en minuscule.
UPPER(column | expr)	Convertit une chaîne de caractères en majuscule.

Fonctions	**Résultats**
LOWER('Cours de SQL')	cours de sql
UPPER ('Cours de SQL')	COURS DE SQL

```
SELECT empno, nom, deptno
FROM emp
WHERE nom = 'blake';

aucune ligne sélectionnée
```

```
SELECT empno, nom, deptno
FROM emp
WHERE nom = UPPER('blake');

EMPNO      NOM        DEPTNO
-----      ---        ------
7698       BLAKE      30
```

3.5 Les fonctions de manipulation de caractères

LENGTH(column | expr)	Permet de récupérer le nombre de caractères d'une chaîne. LENGTH retourne une valeur de type NUMBER.
SUBSTR(column | expr, m [,n])	Permet d'extraire une chaîne de caractères de la chaîne de caractère column (ou issue de expr) sur une longueur n à partir de la position m.
INSTR(column | expr, c)	Permet de récupérer la position de la première occurrence du caractère c dans la chaîne de caractères column ou issue de expr.
LPAD(column | expr, n, 'string')	Permet de placer (n-len(column|expr)) caractères de type string à gauche de la valeur de column (ou expr). Ex : SELECT LPAD('t',4,'X') FROM DUAL; → XXXt
RPAD(column | expr, n, 'string')	Permet de placer (n-len(column|expr)) caractères de type string à droite de la valeur de column (ou expr). Ex : SELECT RPAD('t',4,'X') FROM DUAL; → tXXX
CONCAT(column1 | expr1, Column2 | expr2)	Permet de concaténer la valeur de la première chaîne à la valeur de la seconde chaîne. (équivalent à l'opérateur "||").
TRIM(leading | trailing | both, trim_character **FROM** trim_source)	Permet de couper les caractères trim_character en entête (leading), en fin (trailing) ou les deux (both) d'une chaîne de caractère trim_source.

Fonctions	**Résultats**
CONCAT('Bon', 'jour')	Bonjour
SUBSTR('Bonjour',1,3)	Bon
LENGTH('Bonjour')	7
INSTR('Bonjour','j')	4
LPAD(sal, 10,'*')	******5000
RPAD(sal, 10,'*')	5000******
TRIM('S' FROM 'SSMITH')	MITH

3.6 Les fonctions opérant sur les nombres

ROUND(column | expr [,n])	Permet d'arrondir une valeur column ou issue de expr à n décimales près.

Si n est positif, l'arrondi se fera après la virgule. Si n est négatif l'arrondi se fera avant la virgule (à la dizaine près par exemple). Par défaut n vaut 0.

TRUNCATE(column | expr [,n])	Permet de tronquer une valeur column ou issue de expr à n décimales près.

Si n est positif, la troncation se fera après la virgule. Si n est négatif la troncation se fera avant la virgule (à la dizaine près par exemple). Par défaut n vaut 0.

MOD(m,n)	Permet de retourner le reste de la valeur de m divisé par n.

3.7 Les fonctions opérant sur les dates

SYSDATE()	Permet de retourner la date et l'heure courante.

LAST_DAY(date)	Trouve la date du dernier jour du mois qui contient *date*.

PERIOD_DIFF(p1,p2)	Retourne le nombre de mois séparant deux périodes (p1 - p2). P1 et p2 doivent être au format YYMM ou YYYYMM. P1 et p2 ne sont pas des dates.

3.8 Les fonctions générales

La fonction IFNULL() permet de substituer les valeurs nulles d'une colonne par une valeur choisie.

```
SELECT nom, IFNULL(mgr, 'Aucun') AS MGR
FROM emp;
NOM    MGR
----   ---
KING   Aucun
```

La fonction IF() fait le travail d'un ordre IF-THEN-ELSE.

```
SELECT IF(1>2,2,3) AS faux;

faux
----
3
```

```
SELECT IF(1<2,'yes','no') AS vrai;

vrai
----
yes
```

Le premier argument est le test à effectuer, le deuxieme ce qui sera retourné si la condition est vrai, et le troisième si si la condition est fausse.

3.9 Imbrication de fonctions

Les fonctions peuvent être imbriquées sur plusieurs niveaux. Les fonctions imbriquées sont évaluées de l'intérieur vers l'extérieur.

```
SELECT IF(PERIOD_DIFF(201701,1712) > 0,
                'p2 plus grand',
                'p1 plus grand')
          AS 'Fonctions Imbriquées';

Fonctions Imbriquées
--------------------
p1 plus grand
```

4 LES FONCTIONS DE GROUPE

4.1 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.

5.1.2 Fonctions de groupe	
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

4.2 Exemples de fonctions de groupe

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 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 IFNULL().

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

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

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 ».

```
SELECT AVG(IFNULL(comm,0)) AS moyenne
FROM emp;

moyenne
----------------
157,142857
```

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

4.3 Créer des groupes de données

4.3.1 Le 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 SELECT 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.

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
ORDER BY AVG(sal) DESC;

DEPTNO     AVG(SAL)
---------  ----------
10         2916,66667
20         2175
30         1566,66667
```

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

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

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

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.

4.3.2 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];
```

La différence entre HAVING et WHERE :

WHERE restreint les enregistrements

HAVING restreint les groupes d'enregistrements

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

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

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

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

JOB        PAYROLL
---------  ----------
ANALYST    6000
MANAGER    8275
```

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 ascendante sur les sommes des salaires.

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.

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.

Les fonctions de groupes peuvent avoir comme argument le résultat d'une fonction de groupe (elles peuvent s'imbriquer)

```
SELECT MAX(AVG(sal))
```

```
FROM emp
GROUP BY deptno;

MAX(AVG(SAL))
-------------
2916,66667
```

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.

4.3.3 La clause UNION

La clause UNION est utilisée pour combiner l'information retrouvée par 2, ou plus, ordres SELECT.

```
SELECT column1, column2
FROM table1
[WHERE condition]
UNION
SELECT column1bis, column2bis
FROM table2
[WHERE condition]
[ORDER BY column1];
```

Les colonnes des jeux de résultats doivent se correspondre. Il n'est pas nécessaire qu'elles portent le même nom, mais chaque jeu de résultats doit avoir le même nombre de colonnes, et ces colonnes doivent avoir des types de données compatibles et être dans le même ordre.
La liste des lignes issues des tables citées dans les SELECT de l'UNION ne comporte aucun doublon, sauf si l'option ALL est spécifiée.
Les noms de colonne du jeu de résultats sont ceux de la première instruction SELECT. Les alias de colonne s'ils sont à définir, seront donc définis dans la première instruction SELECT.
S'il y a une clause ORDER BY, elle sera définie à la suite de la dernière instruction SELECT.

```
SELECT ename "Nom"
FROM cadre
WHERE deptno = 40
UNION
SELECT ename
FROM emp
WHERE deptno = 40
ORDER BY "Nom";

Nom
-------------
...
```

Cette instruction composée de 2 SELECT retourne la liste triée des noms des cadres et des employés du département 40.

5 AFFICHER DES DONNEES ISSUES DE PLUSIEURS TABLES

5.1 Les jointures

5.1.1 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.

Pour représenter la jointure il y a 2 méthodes :
- La méthode ensembliste qui réalise l'intersection de deux ensembles (Sous-requêtes)
- La méthode prédicative qui vérifie l'égalité de deux attributs.

Il existe quatre types de jointures :
- Equi-jointure (equijoin)
- Non equi-jointure (non-equijoin)
- Jointure externe (outer join)
- Auto jointure (self join)

5.1.2 La méthode prédicative

Il y a un seul SELECT pour toute la requête.
La liste de toutes les tables concernées apparaît dans le FROM.
La traduction de la jointure se fait par l'équation de jointure (égalité entre 2 attributs).
Il existe deux méthodes pour exprimer les conditions de jointure.
Une première méthode qui consiste à exprimer les conditions de jointure dans la clause WHERE.

```
SELECT table1.column3, table2.column3
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

Une deuxième méthode consiste à exprimer les jointures dans la clause FROM.

```
SELECT table1.column3, table2.column3
FROM table1
JOIN table2 ON table1.column1 = table2.column2;
```

5.1.3 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.

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

5.2 Récupérer des enregistrements 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.
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 emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
FROM emp, dept
WHERE emp.deptno = dept.deptno;
 OU
SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
FROM emp
INNER JOIN dept ON emp.deptno = dept.deptno;

EMPNO      ENAME      DEPTNO DEPTNO   LOC
------     ------     ------ ------   ------
7782       CLARK      10         10   NEW YORK
7839       KING       10         10   NEW YORK
7934       MILLER     10         10   NEW YORK
7369       SMITH      20         20   DALLAS
7876       ADAMS      20         20   DALLAS
```

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.

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

```
SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
FROM emp
INNER JOIN dept ON emp.deptno = dept.deptno
WHERE emp.ename = 'KING';

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

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

```
SELECT e.empno, e.ename, e.deptno, d.deptno, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno
AND e.ename = 'KING';
```

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.

5.3 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.

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

ENAME SAL  GRADE
----- ---- -----
SMITH 800  1
ADAMS 1100 1
JAMES 950  1
FORD  3000 4
KING  5000 5
```

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

Aucune limite des tranches de salaire dans la table SALGRADE ne se chevauche.

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.

5.4 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.
On utilise une clause LEFT, RIGHT ou FULL OUTER JOIN.
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.

```
SELECT table1.column, table2.column
FROM table1
LEFT JOIN table2 ON table1.column = table2.column;
```

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

```
SELECT table1.column, table2.column
FROM table1
RIGHT JOIN table2 ON table1.column = table2.column;
```

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

```
SELECT table1.column, table2.column
FROM table1
FULL JOIN table2
ON table1.column = table2.column;
```

Cette requête affiche tous les enregistrements des 2 tables même si ils ne respectent pas la condition de jointure.

5.5 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 tbalias1.column, tbalias2.column
FROM table1 AS tbalias1, table1 AS tbalias2
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 (tbalias1, tbalias2).

```
SELECT worker.ename||' travaille pour '||manager.ename AS 'CHEFS'
FROM emp worker, emp manager
WHERE worker.mgr = manager.empno;

CHEFS
------------------------------
SCOTT travaille pour JONES
ALLEN travaille pour BLAKE
```

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

6 LES SOUS-REQUETES

6.1 Règles d'écriture

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 connue. 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, on place une requête à l'intérieur d'une autre.
La la sous-requête retourne une valeur qui est utilisée par la 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 ename
FROM emp
WHERE sal > (SELECT sal
             FROM emp
             WHERE ename = 'JONES');
ENAME
----------
SCOTT
KING
FORD
```

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.

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

6.2 Sous-requêtes 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 :
<, >, =, <=, >=, <>.

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

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job
             FROM emp
             WHERE empno = 7369)
AND sal >   (SELECT sal
             FROM emp
             WHERE empno = 7876);

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

Des fonctions de groupes peuvent être utilisées dans une sous-requête pour opérer sur un groupe de valeurs. (cf. "Les fonctions de groupe").
Placer dans la sous-requête, elles permettent de ne retourner qu'une seule valeur à la requête principale.

Les sous-requêtes peuvent être utilisées dans la clause HAVING :

```
SELECT deptno, MIN(sal)
FROM emp
GROUP BY deptno
HAVING MIN(sal) > (SELECT MIN(sal)
                   FROM emp
                   WHERE deptno = 20);
DEPTNO MIN(SAL)
------ --------
10     1300
30     950
```

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

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 bien qu'un seul enregistrement.

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

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job
             FROM emp
             WHERE ename='SMYTHE');

aucune ligne sélectionnée
```

Cette requête n'affiche aucun résultat. Aucun 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.

6.3 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.

Une sous-requête multiple-row ne peut être utilisée qu'avec les opérateurs de comparaison multiple-row : IN, NOT IN, ANY, ALL, BETWEEN et EXISTS.

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.

```
SELECT ename, sal, deptno
FROM emp
WHERE sal IN (SELECT MIN(sal)
              FROM emp
              GROUP BY deptno);

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

Cette requête affiche les employés dont le salaire est égal au salaire minimum d'un département.

L'opérateur **ANY** compare une valeur à chaque valeur retournée par la sous-requête.
< ANY signifie **plus petit que la valeur la plus grande.**
> ANY signifie **plus grand que la valeur la plus petite**.
= ANY est équivalent à IN.
L'opérateur NOT ne pas peut être utilisé avec l'opérateur ANY.

```
SELECT empno, ename, job
FROM emp
WHERE sal < ANY (SELECT sal
                 FROM emp
                 WHERE job = 'CLERK')
AND job <> 'CLERK';

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

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 d'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 **ALL** compare une valeur à toutes les valeurs retournées par la sous-requête.
> ALL signifie **plus grand que la valeur la plus grande**.
< ALL signifie **plus petit que la valeur la plus petite**.
L'opérateur NOT ne peut être utilisé avec l'opérateur ALL.

```
SELECT empno, ename, job
FROM emp
WHERE sal > ALL (SELECT AVG(sal)
                 FROM emp
                 GROUP BY deptno);

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

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.

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

6.4 Les 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.

6.4.1 Comparaisons de plusieurs colonnes

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

Comparaison "pairwise" :

```
SELECT ename, deptno, sal, comm
FROM emp
WHERE (sal, IFNULL(comm,-1)) IN (SELECT sal, IFNULL(comm,-1)
                                 FROM emp
                                 WHERE deptno = 30)
AND deptno <> 30;

aucune ligne sélectionnée
```

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 employé ne faisant pas parti du département 30 n'a le même salaire et la même commission qu'un employé du département 30.

Comparaison non "pairwise" :

```
SELECT ename, deptno, sal, comm
FROM emp
WHERE sal IN (SELECT sal
              FROM emp
              WHERE deptno = 30)
AND IFNULL(comm,-1) IN (SELECT IFNULL(comm,-1)
              FROM emp
              WHERE deptno = 30)
AND deptno <> 30 ;

ENAME DEPTNO SAL  COMM
----- ------ ---  -----
CLARK 10     1500 300
```

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 retourne un seul enregistrement : CLARK.
En effet CLARK a le même salaire que TURNER et la même commission qu' ALLEN.

6.4.2 Sous-requêtes multiple-column dans la clause FROM

Des sous-requêtes multiple-column peuvent être écrites dans la clause FROM. La méthode est similaire quel que 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.

```
SELECT a.ename, a.sal, a.deptno, b.salavg
FROM emp a, (SELECT deptno, AVG(sal) AS salavg
                FROM emp
             GROUP BY deptno) AS b
WHERE a.deptno = b.deptno
AND a.sal < b.salavg;

ENAME   SAL   DEPTNO  SALAVG
-----   ----  ------  -------
CLARK   1500  10      2600
MILLER  1300  10      2600
SMITH   800   20      2175
```

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