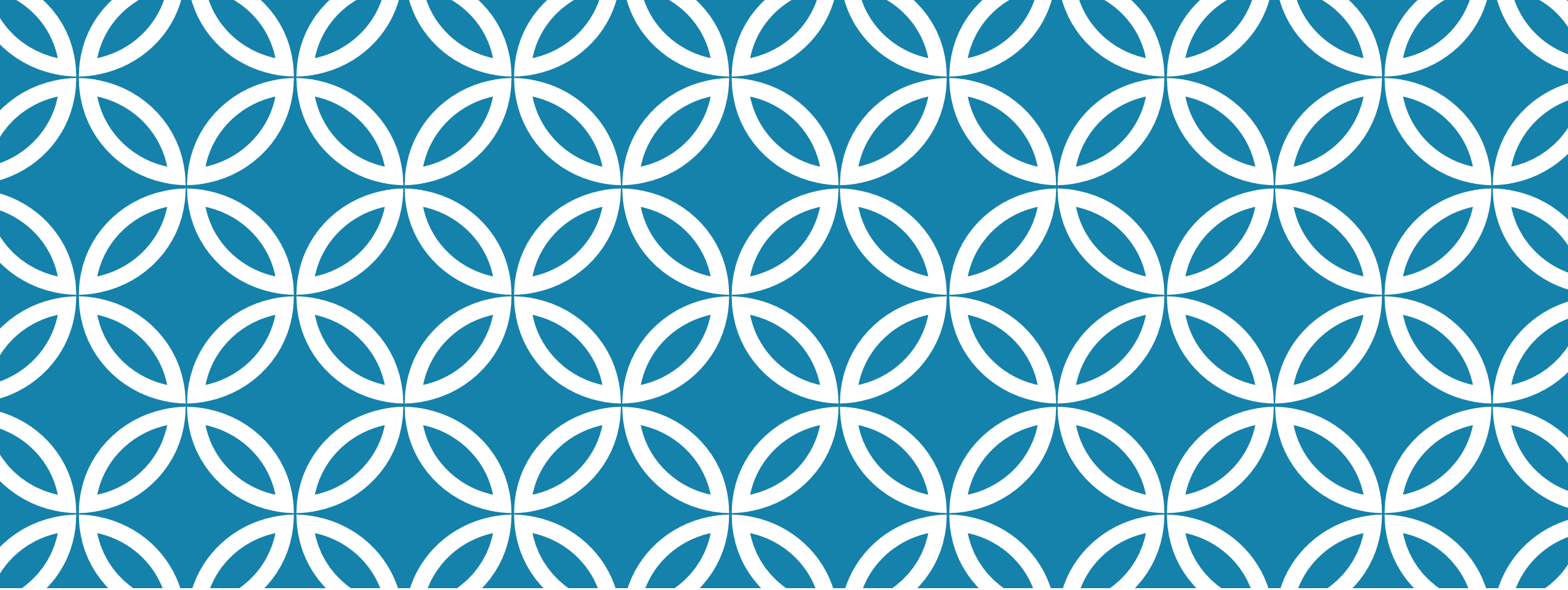
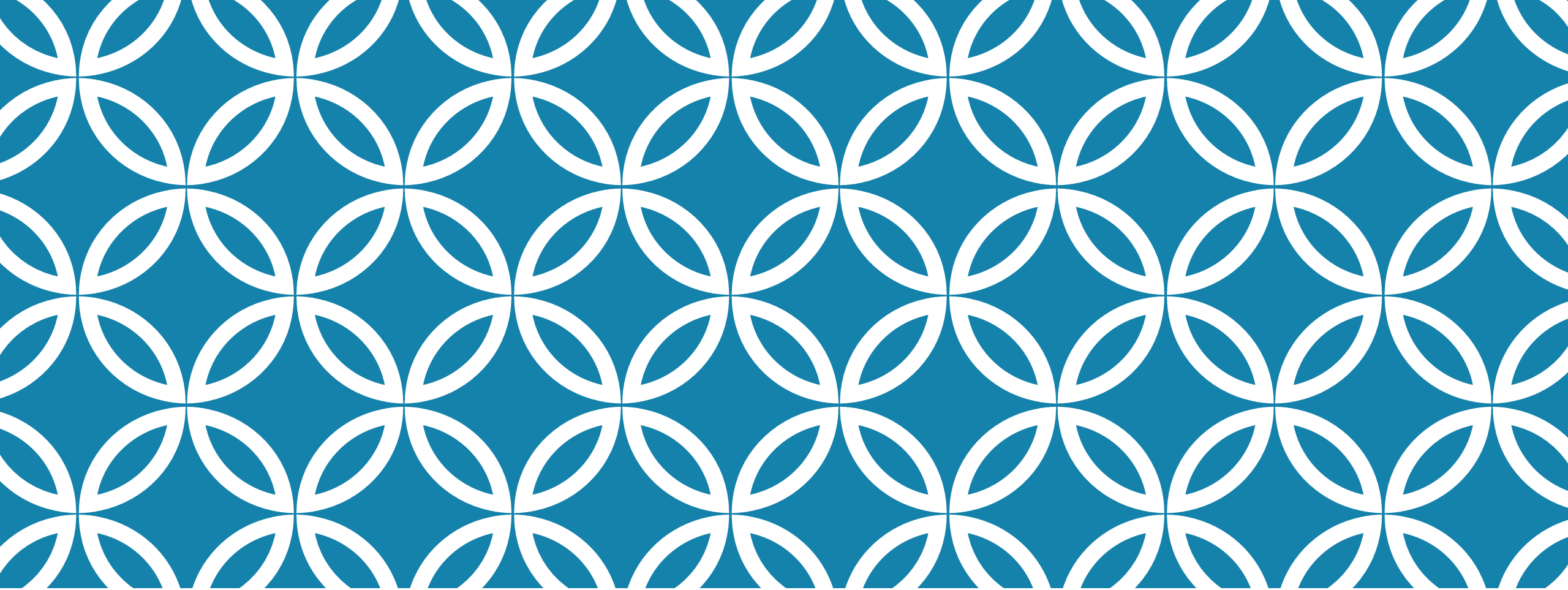


FORMATION SQL

Jordan ABID



BASE DE DONNÉES

CONCEPTS D'UNE BASE DE DONNÉES RELATIONNELLE

Toute organisation a besoin d'informations.

Exemples :

Une bibliothèque maintient une liste de membres, de livres...

Une société maintient des informations sur ses salariés, ses départements... Ces informations sont appelées des données.

Elles peuvent être stockées sur différents types de support et format comme : des fiches papiers, des tableaux Excel...ou des bases de données.

STRUCTURE GÉNÉRALE D'UNE TABLE :

Un ligne, un enregistrement ou un tuple (row) :
Ensemble de caractéristiques définissant une occurrence de l'objet table. (Correspond dans table EMP aux informations relatives a un employé). L'ordre des lignes stockées dans la table est sans importance car un ordre de tri peut être spécifie lors d'une requête SQL.

Une colonne (column) :
Ensemble de données relatives à une information caractéristique (dans la table EMP, tous les employés ont un nom, un salaire...). Une colonne peut contenir une clé primaire, une clé etrangère ou des valeurs simples.

Une clé primaire (Primary Key PK) :
Attribut d'une table permettant d'identifier un enregistrement de manière unique. Cette valeur ne peut donc pas être nulle ou double. (En règle générale, cette valeur n'est pas modifiée par l'utilisateur).

Une clé etrangère (Foreign Key FK) :
Attributs référençant la clé d'une autre table. Une clé étrangère traduit une relation entre deux tables.

Un champ (field) :
Intersection d'une ligne avec une colonne. Un champ ne peut contenir qu'une seule valeur . Sa valeur peut être nulle (elle ne contiendra aucune valeur).

Les valeurs d'une clé étrangère correspondent aux valeurs d'une clé primaire dans une autre table.

Les colonnes qui ne sont ni clé primaire ni clé étrangère contiennent des valeurs qui ne font pas référence à des valeurs d'une autre table.

SQL

Le **SQL** (**S**tructured **Q**uery **L**anguage) est un langage de programmation qui permet de récupérer et manipuler les données dans une base de données relationnelle.

Il existe cinq types d'ordre SQL :

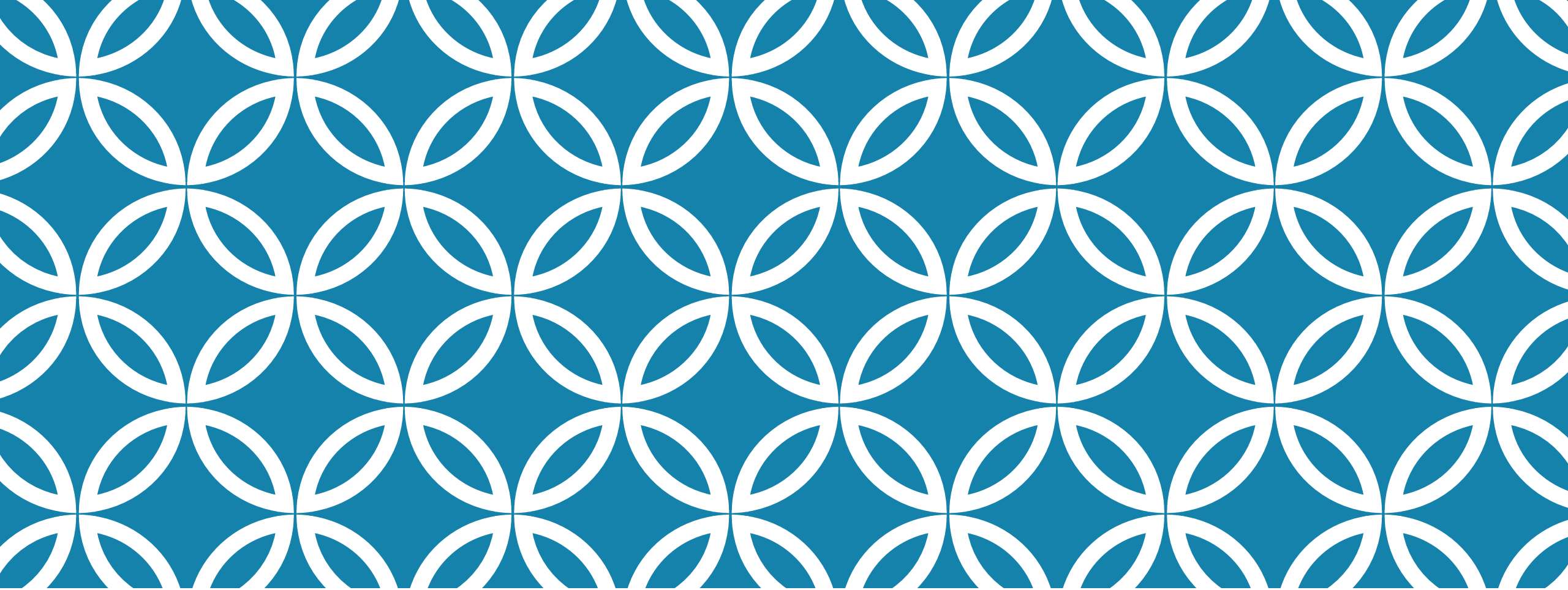
Data **R**etrieval **L**angage (DRL) :
Ensemble de commandes qui permettent de récupérer les données contenues dans une ou plusieurs table de la base. (Exemple : l'ordre SELECT)

Data **M**anipulation **L**angage (DML) :
Ensemble de commandes qui permettent de modifier les données de la base. (Exemple : les ordres INSERT, DELETE, UPDATE)

Data **D**efinition **L**angage (DDL) :
Ensemble de commandes qui permettent de modifier la structure de la base. (Exemple : les ordres CREATE, DROP, ALTER, RENAME)

Transaction **C**ontrol **S**tatement (TCS) :
Ensemble de commandes qui permettent d'administrer les changement effectués par les commandes DML. (Exemple : les commandes COMMIT, ROLLBACK, SAVEPOINT)

Data **C**ontrol **L**anguage (DCL) :
Est un ensemble de commandes qui permettent de contrôler les accès utilisateur à la base de données. (Exemple : les ordres GRANT, REVOKE)



SÉLECTION DES ENREGISTREMENTS

Tables Utilisées dans le Cours

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	1500		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
			7782	23-JAN-82			10

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

ECRITURE D'ORDRES SQL BASIQUES

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

SELECT *« quoi ? »*

FROM *« quelle 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 données

Exemples :

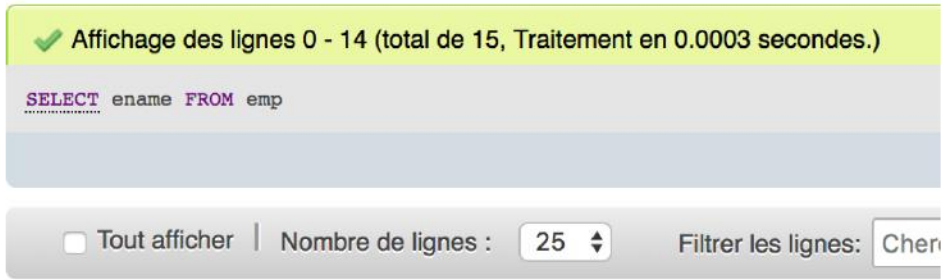
1) *SELECT ename FROM emp;*

2) *SELECT job,sal,sal *12 FROM emp;*

3) *SELECT * FROM emp;*

*Le caractère ' * ' signifie que toutes les colonnes sont sélectionnées.*

ECRITURE D'ORDRES SQL BASIQUES



+ Options

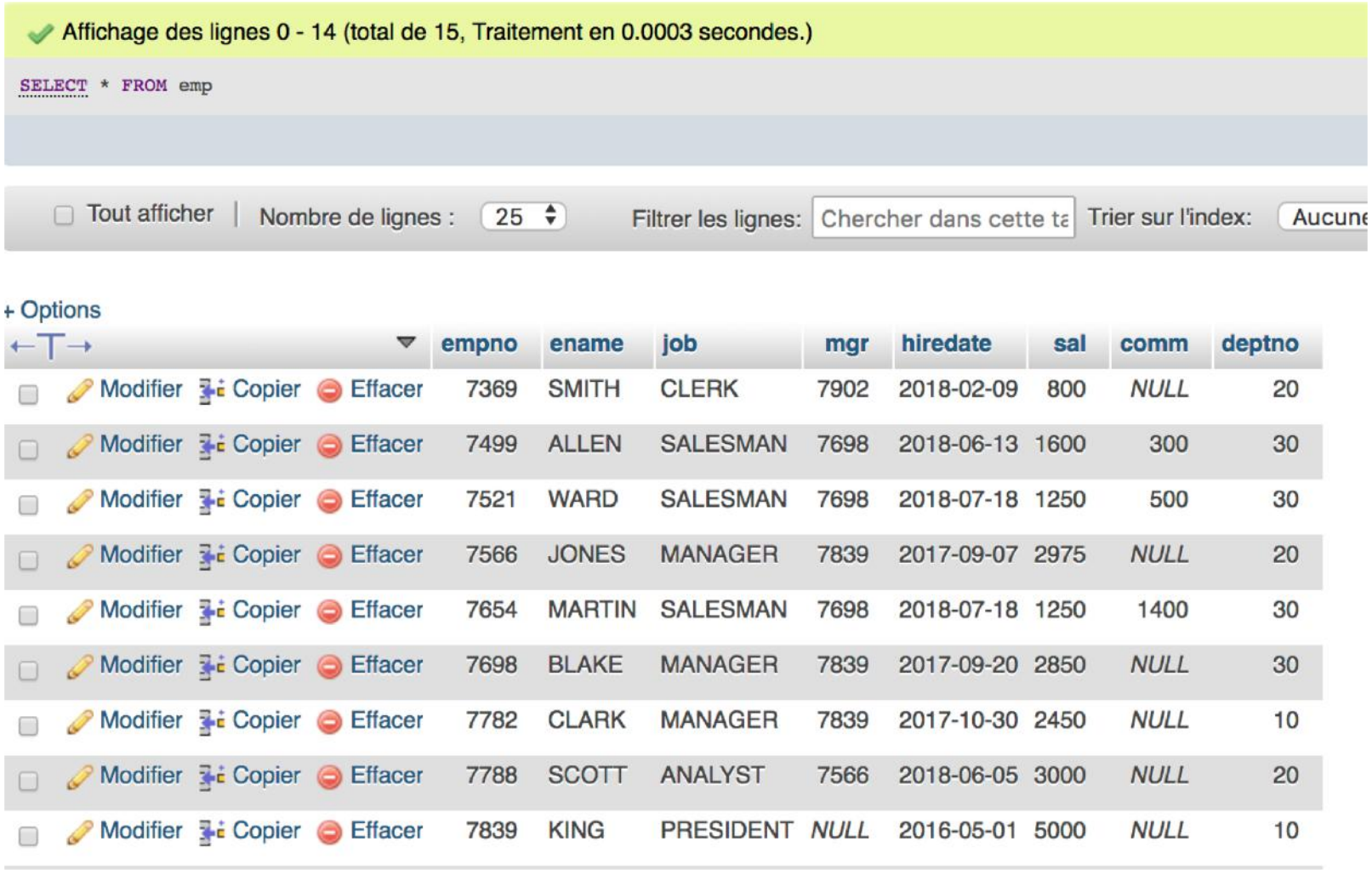
ename
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
WILKIN

ECRITURE D'ORDRES SQL BASIQUES

+ Options

job	sal	sal*12
CLERK	800	9600
SALESMAN	1600	19200
SALESMAN	1250	15000
MANAGER	2975	35700

ÉCRITURE D'ORDRES SQL BASIQUES



Affichage des lignes 0 - 14 (total de 15, Traitement en 0.0003 secondes.)

```
SELECT * FROM emp
```

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette ta Trier sur l'index: Aucune

+ Options

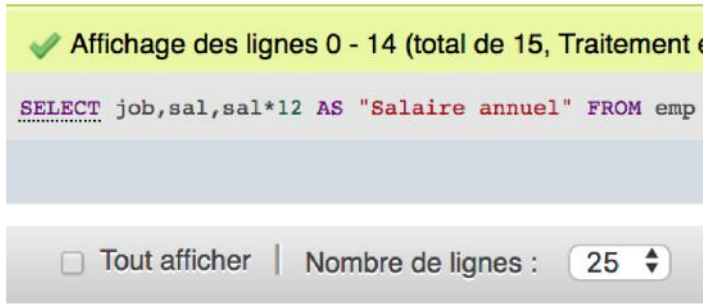
		empno	ename	job	mgr	hiredate	sal	comm	deptno
☐	Modifier Copier Effacer	7369	SMITH	CLERK	7902	2018-02-09	800	*NULL*	20
☐	Modifier Copier Effacer	7499	ALLEN	SALESMAN	7698	2018-06-13	1600	300	30
☐	Modifier Copier Effacer	7521	WARD	SALESMAN	7698	2018-07-18	1250	500	30
☐	Modifier Copier Effacer	7566	JONES	MANAGER	7839	2017-09-07	2975	*NULL*	20
☐	Modifier Copier Effacer	7654	MARTIN	SALESMAN	7698	2018-07-18	1250	1400	30
☐	Modifier Copier Effacer	7698	BLAKE	MANAGER	7839	2017-09-20	2850	*NULL*	30
☐	Modifier Copier Effacer	7782	CLARK	MANAGER	7839	2017-10-30	2450	*NULL*	10
☐	Modifier Copier Effacer	7788	SCOTT	ANALYST	7566	2018-06-05	3000	*NULL*	20
☐	Modifier Copier Effacer	7839	KING	PRESIDENT	*NULL*	2016-05-01	5000	*NULL*	10

ALIAS DE COLONNE

Un alias de colonne est une chaîne de caractère qui se substitut au nom de la colonne pour le traitement et l'affichage de la colonne.

SELECT *column1* **AS "***alias1"*, *column2* **AS "***alias2"*...

FROM *table*;



- Options

job	sal	Salaire annuel
CLERK	800	9600
SALESMAN	1600	19200
SALESMAN	1250	15000
MANAGER	2975	35700
SALESMAN	1250	15000
MANAGER	2850	34200
MANAGER	2450	29400
ANALYST	3000	36000

OPÉRATEUR DE CONCATÉNATION

La combinaison de caractères " **||** " est utilisée pour concaténer des colonnes ou des chaines de caractères à d'autres colonnes.

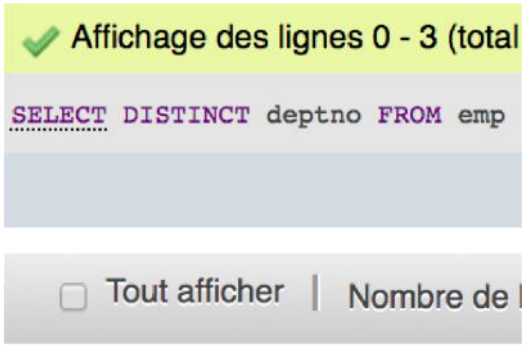
SELECT *ename* || *job* **AS** *"Password"* **FROM** *emp*;

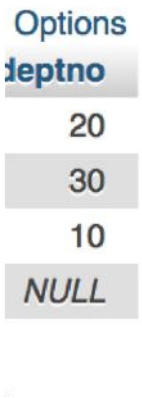
Avec MYSQL :
SELECT CONCAT(*ename,job*) **AS** *"Password"* **FROM** *emp*;

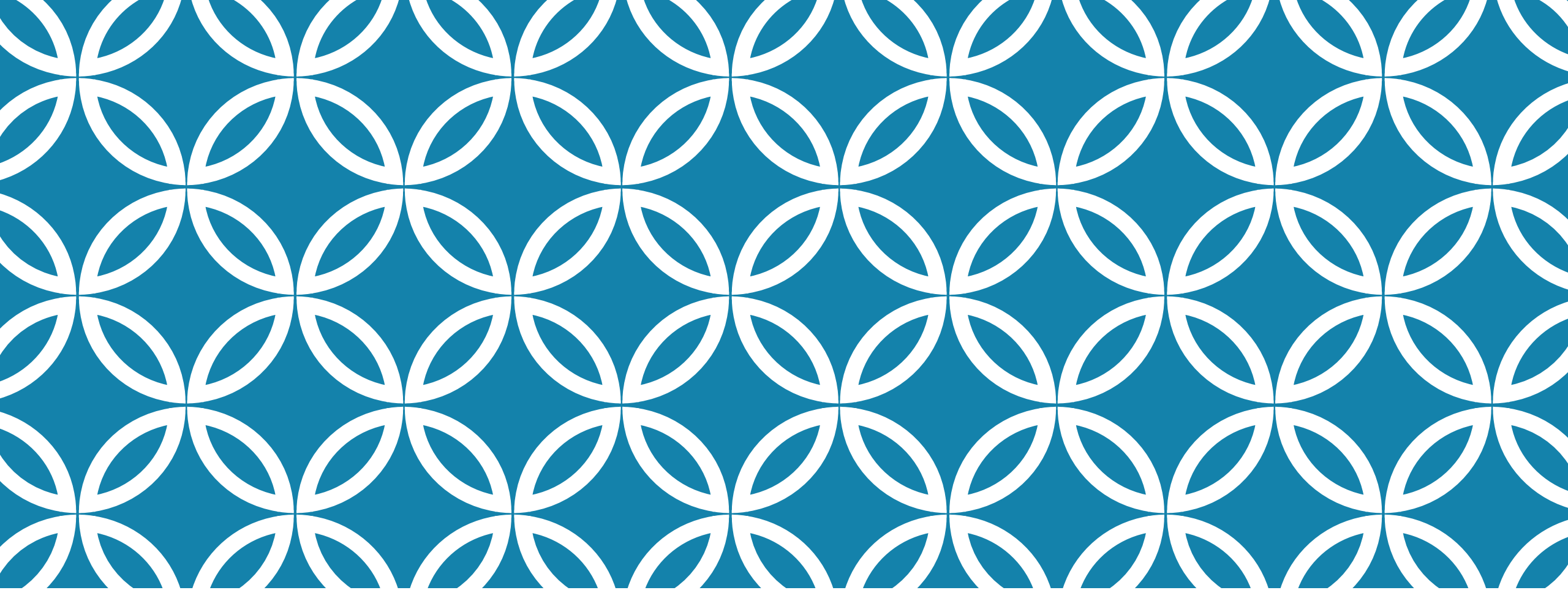
Password
SMITHCLERK
ALLENSALESMAN
WARDSALESMAN
JONESMANAGER
MARTINSALESMAN
BLAKEMANAGER
CLARKMANAGER
SCOTTANALYST
KINGPRESIDENT
TURNERSALESMAN
ADAMSCLERK
JAMESCLERK
FORDANALYST
MILLERCLERK
WILKINCLERK

ELIMINATION DES DOUBLONS

Le mot-clé **DISCTINCT** élimine les doublons dans le résultat de la requête lors de l'affichage. Un doublon est un enregistrement qui se répète plusieurs fois.







RESTREINDRE DES ENREGISTREMENTS

LA CLAUSE WHERE

La clause WHERE restreint la requête aux enregistrements qui respectent sa ou ses conditions.

SELECT [**DISTINCT**] { * | *column* [*alias*], *expr*, ...}
FROM *table*
[**WHERE** *colonne operateur valeur*] ;

```
SELECT ename, job, deptno
FROM emp
WHERE job = "ANALYST";
```

```
SELECT ename, job, deptno
FROM emp
WHERE deptno != 10;
```

LES OPÉRATEURS DE COMPARAISON

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

LES OPÉRATEURS LOGIQUES

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

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

```
SELECT ename,sal,deptno
FROM  emp
WHERE sal > 1000 and sal<3500
        and (deptno=10 or deptno=20);
```

ename	sal	deptno
JONES	2975	20
CLARK	2450	10
SCOTT	3000	20
ADAMS	1100	20
FORD	3000	20
MILLER	1300	10

L'OPÉRATEUR BETWEEN

```
SELECT ename,sal,deptno
FROM  emp
WHERE sal BETWEEN 1000 and 3500
            and (deptno=10 or deptno=20);
```

ename	sal	deptno
JONES	2975	20
CLARK	2450	10
SCOTT	3000	20
ADAMS	1100	20
FORD	3000	20
MILLER	1300	10

L'OPÉRATEUR IN

```
SELECT ename,sal,deptno
FROM  emp
WHERE sal BETWEEN 1000 and 3500
        and deptno IN(10,20);
```

ename	sal	deptno
JONES	2975	20
CLARK	2450	10
SCOTT	3000	20
ADAMS	1100	20
FORD	3000	20
MILLER	1300	10

L'OPÉRATEUR LIKE

L'opérateur **LIKE** permet de faire des recherches de caractères spécifiques dans une chaîne de caractères données.

SELECT ename

FROM emp

WHERE ename **LIKE** "_A%"

ename
WARD
MARTIN
JAMES

Le symbole '_' représente un seul caractère quelconque.

Le symbole '%' représente une série de zéros ou de caractères.

L'OPÉRATEUR IS NULL

L'opérateur **IS NULL** permet d'afficher les enregistrements dont certains champs contiennent des valeurs nulle.

Une valeur nulle signifie que la valeur n'est pas disponible, non assignée, inconnue ou inapplicable.

SELECT ename,mgr

FROM emp

WHERE mgr **IS NULL**

ename	mgr
KING	*NULL*

L'OPERATEUR NOT

L'opérateur NOT permet d'inverser les autres opérateurs :

- **WHERE** *deptno* **NOT IN** *(30,40)*
- **WHERE** *ename* **NOT LIKE '_A%'**
- **WHERE** *sal* **NOT BETWEEN** *1000* **AND** *3000*
- **WHERE** *com* **IS NOT NULL**

LA CLAUSE ORDER BY

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

SELECT [**DISTINCT**] { * | {*column* [*alias*] | *expr*, ...}}

FROM *table*

[**WHERE** *condition(s)*];
[**ORDER BY**{*column* | *alias*} [**ASC** | **DESC**] ;

```
SELECT ename FROM emp ORDER BY ename
```

ename ▲ 1
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
WILKIN

```
SELECT ename FROM emp ORDER BY ename DESC
```

ename ▼ 1
WILKIN
WARD
TURNER
SMITH
SCOTT
MILLER
MARTIN
KING
JONES
JAMES
FORD
CLARK
BLAKE
ALLEN
ADAMS

TRIER SUR PLUSIEURS COLONNES

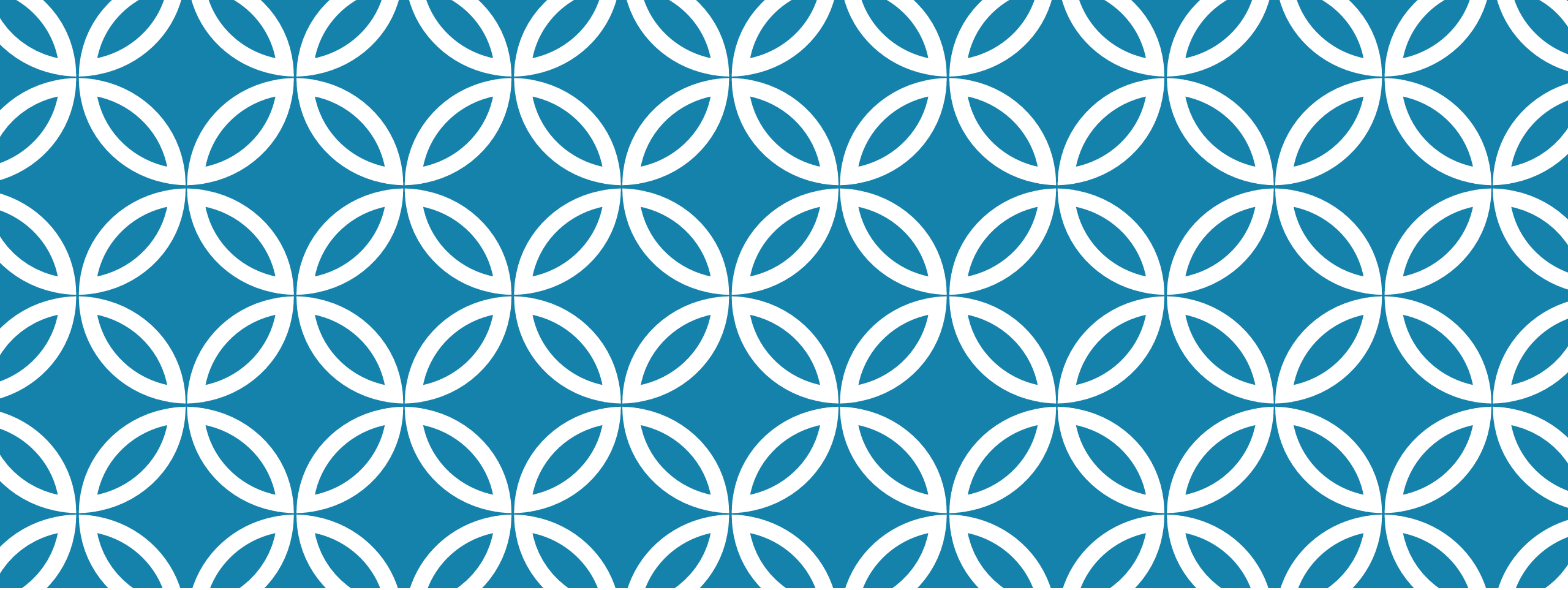
Les enregistrements peuvent être triés sur plusieurs colonnes.

ORDER BY *column1*, *column2* ;
ORDER BY *column1* [**DESC** | **ASC**], *column2* [**DESC** | **ASC**];

Column1 et *column2* peuvent être des noms de colonnes, des expressions, des alias ou des numéros de colonnes.

```
SELECT deptno,ename FROM emp ORDER BY deptno DESC, ename
```

deptno 1	ename 2
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
10	CLARK
10	KING
10	MILLER
NULL	WILKIN



LES FONCTIONS SQL

LES FONCTIONS DE CONVERSION DE CASSE

Fonctions	Résultats	Fonctions	Résultats
INITCAP(colonne)	Convertit la première lettre de chaque mot d'une chaîne de caractères en majuscule et les autres lettres en minuscule.	INITCAP('Cours de SQL')	Cours De Sql
LOWER(colonne)	Convertit une chaîne de caractères en minuscule.	LOWER('Cours de SQL')	cours de sql
UPPER(colonne)	Convertit une chaîne de caractères en majuscule.	UPPER ('Cours de SQL')	COURS DE SQL

LES FONCTIONS DE MANIPULATION DE CARACTÈRES

Fonctions	Résultats	Fonctions	Résultats
LENGTH(colonne)	Permet de récupérer le nombre de caractères d'une chaîne. LENGTH retourne une valeur de type NUMBER.	LENGTH ('Bonjour')	7
SUBSTR(colonne,m,n)	Permet d'extraire une chaîne de caractères de la chaîne de caractère colonne (ou issue de expr) sur une longueur n à partir de la position m.	SUBSTR ('Bonjour',1,3)	Bon
INSTR(colonne,c)	Permet de récupérer la position de la première occurrence du caractère c dans la chaîne de caractères colonne ou issue de expr.	INSTR('Bonjour','j')	4
LPAD(colonne,n,"string") **RPAD**(colonne,n,"string")	Permet de placer n caractères de type string à gauche/droite de la valeur de colonne.	LPAD(sal,8,"*") RPAD(sal,8,"*")	*****850 850*****

LES FONCTIONS OPÉRANTS SUR LES NOMBRES

Fonctions	Résultats	Fonctions	Résultats
ROUND (colonne [,n])	Permet d'arrondir une valeur colonne ou issue de expr à n décimales près.	ROUND (98.6)	99
TRUNC(colonne [,n])	Permet de tronquer une valeur colonne ou issue de expr à n décimales près.	TRUNC (98.6)	98

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.

Les fonctions ROUND et TRUNC peuvent-être utilisées avec des dates

OPÉRATIONS SUR LES DATES

Le format interne à la base (Internal format) est : century, year, month, day, hour, minutes, seconds

L'affichage par défaut est DD-MON-YY soit par exemple 14-JUI-80

La table DUAL peut être utilisée pour afficher la date du jour :

Opération	Résultat	Description
date + number	date	ajoute un nombre de jours à une date
date - number	date	soustrait un nombre de jours à une date
date - date	nombre de jours	soustrait une date à une autre date
date + number/24	date	ajoute un nombre d'heures à une date

LES FONCTIONS SUR LES DATES

Fonctions	Résultats	Fonctions	Résultats
MONTHS_BETWEEN(date1,date2)	Retourne le nombre de mois séparant deux dates. Le résultat peut-être positif ou négatif.	MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
TIMESTAMPDIFF(YEAR,date1,date2) (MYSQL)	Retourne le nombre d'années séparant deux dates (YEAR/MONTH/DAY…)	TIMESTAMPDIFF (MONTH,'1995-05-01','1996-05-08')	12
ADD_MONTHS(date,n)	Ajoute n mois à une date. n doit être un entier positif ou négatif.	ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY(date, 'day of week')	Trouve la date du prochain jour de la semaine (day of week) suivant date. La valeur de day of week doit être un nombre représentant le jour ou une chaîne de caractères.	NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY(date)	Trouve la date du dernier jour du mois qui contient date.	LAST_DAY ('01-SEP-95')	'30-SEP-95'
ROUND(date [,'format'])	Retourne date arrondie à l'unité spécifié par format. Si le format est omis, date est arrondie au jour le plus près.	ROUND ('25-JUL-95','MONTH')	'01-AUG-95'
TRUNC(date [,'format'])	Retourne date tronquée à l'unité spécifié par format. Si le format est omis, date est tronquée au jour le plus près.	TRUNC ('25-JUL-95','YEA R')	'01-JAN-95'

LES FONCTIONS DE CONVERSION

Voici les trois principales fonctions de conversion explicite de types de données :

TO_CHAR(number I date [,'format']) : convertit un nombre ou une date en une chaîne de caractères

TO_NUMBER(char ['format']) : convertit une chaîne de caractères en un nombre

TO_DATE(char [,'format']) : convertit une chaîne de caractères en une date.

LA FONCTION TO_CHAR AVEC DES DATES

Voici les différents formats de conversion de la fonction **TO_CHAR** avec des dates :

YYYY	Æ année sur quatre chiffres
YEAR	Æ année écrite en toutes lettres
MM	Æ le mois sur deux caractères
MONTH	Æ le mois en toutes lettres
DY	Æ le jour de la semaine en trois lettres
DAY	Æ le jour de la semaine en toutes lettres
WW ou W	Æ semaine de l'année ou du mois
DDD	Æ jour de l'année
DD	Æ jour du mois
D	Æ jour de la semaine
J	Æ le nombre de jour depuis le 31 décembre 4713 BC
Q	Æ quart de l'année
MON	Æ le mois sur trois caractères
RM	Æ numéro romain du mois
CC	Æ siècle
fmDAY	Æ supprime les espaces
AM ou PM	Æ indicateur de méridien
HH ou HH12 ou HH24	Æ heure du jour
MI	Æ minutes (0-59)
SS	Æ secondes (0-59)
SSSS	Æ secondes (0-86399)
TH	Æ nombre ordinal
SP	Æ nombre écrit en toutes lettres
SPTH ou THSP	Æ nombre ordinal écrit en toutes lettres

LA FONCTION TO_CHAR AVEC DES DATES

```
SQL> SELECT    ename,
  2            TO_CHAR(hiredate, 'fmDD Month YYYY') HIREDATE
  3  FROM      emp;

ENAME      HIREDATE
---------- -----------------
KING       17 November 1981
BLAKE      1 May 1981
CLARK      9 June 1981
JONES      2 April 1981
MARTIN     28 September 1981
ALLEN      20 February 1981
...
14 rows selected.
```

Le format doit être entouré de simples côtes.

Le format est sensible à la casse. Il doit inclure des éléments de format de date valides.

Pour éliminer les blancs ou supprimer les zéros, il faut utiliser l'élément "fill mode".

LA FONCTION NVL / IFNULL

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

NVL (expr1,expr2) :

expr1 : valeur source ou expression pouvant contenir une valeur nulle

expr2 : valeur de substitution

expr1 et *expr2* doivent être du même type de données.

```
SELECT ename,sal,comm,sal+comm as "salaire total" from emp
```

ename	sal	comm	salaire total
SMITH	800	*NULL*	*NULL*
ALLEN	1600	300	1900
WARD	1250	500	1750
JONES	2975	*NULL*	*NULL*
MARTIN	1250	1400	2650
BLAKE	2850	*NULL*	*NULL*
CLARK	2450	*NULL*	*NULL*
SCOTT	3000	*NULL*	*NULL*
KING	5000	*NULL*	*NULL*
TURNER	1500	0	1500
ADAMS	1100	*NULL*	*NULL*
JAMES	950	*NULL*	*NULL*
FORD	3000	*NULL*	*NULL*
MILLER	1300	*NULL*	*NULL*
WILKIN	1280	*NULL*	*NULL*

```
SELECT ename,sal,comm,sal+IFNULL(comm,0) as "salaire total" from emp
```

ename	sal	comm	salaire total
SMITH	800	*NULL*	800
ALLEN	1600	300	1900
WARD	1250	500	1750
JONES	2975	*NULL*	2975
MARTIN	1250	1400	2650
BLAKE	2850	*NULL*	2850
CLARK	2450	*NULL*	2450
SCOTT	3000	*NULL*	3000
KING	5000	*NULL*	5000
TURNER	1500	0	1500
ADAMS	1100	*NULL*	1100
JAMES	950	*NULL*	950
FORD	3000	*NULL*	3000
MILLER	1300	*NULL*	1300
WILKIN	1280	*NULL*	1280

LA FONCTION DECODE

La fonction DECODE peut faire le travail d'un ordre IF-THEN-ELSE ou d'un ordre CASE.

DECODE (*colonne*,

search1, *result1*
[, *search2*, *result2*,]

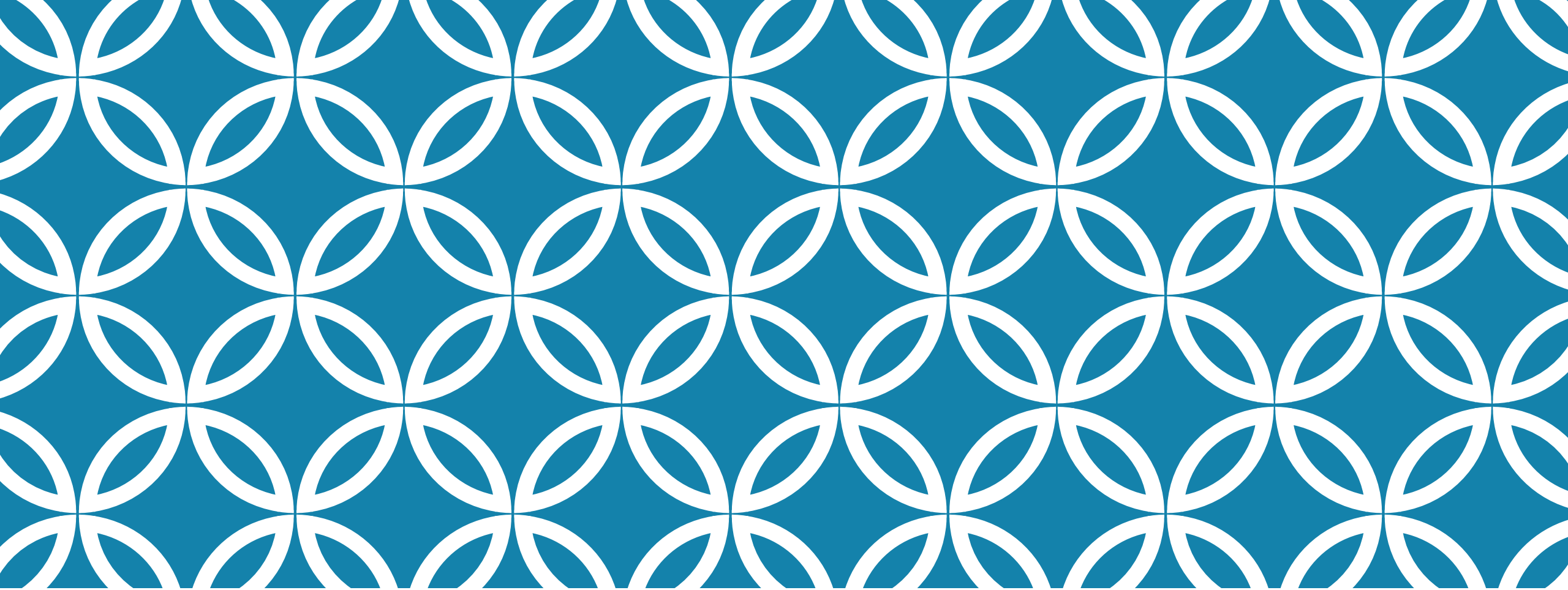
[*search3*, *result3*,] [...,...]
[*résultat par défaut*])

La fonction DECODE décode l'expression après l'avoir comparé à chaque valeur *search*.

Si l'expression est la même que *search*, la valeur *result* est retournée. Si la valeur par défaut est omis et qu'aucune valeur *search* ne correspond à l'expression, une valeur nulle est retournée.

LA FONCTION DECODE

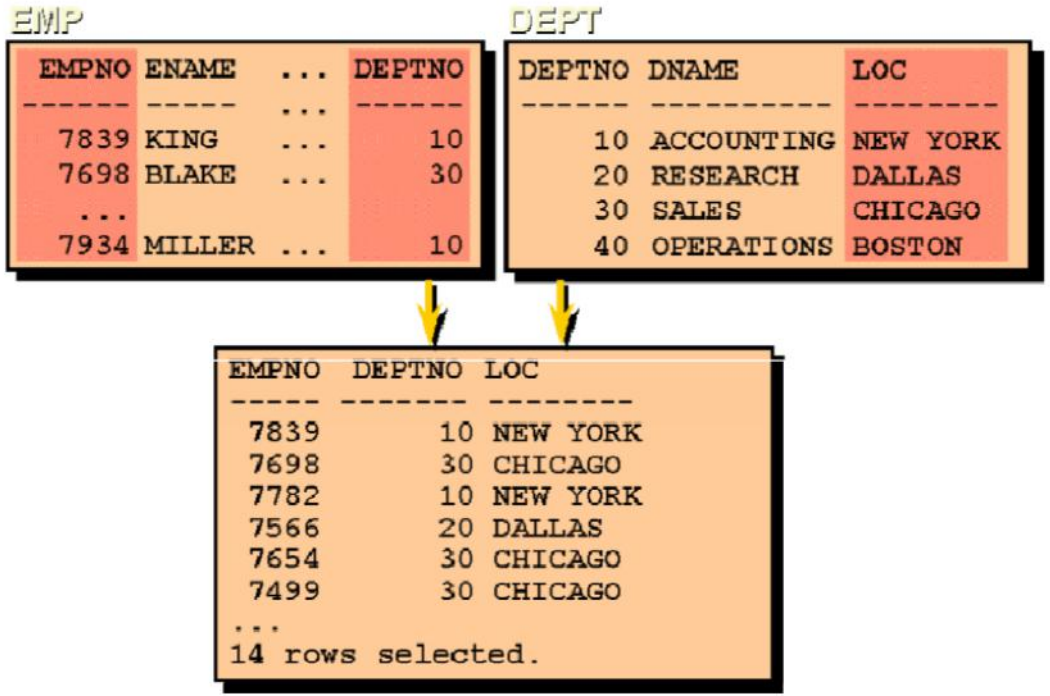
```
SQL> SELECT    job, sal,
  2            DECODE(job,'ANALYST', SAL*1.1,
  3                       'CLERK',   SAL*1.15,
  4                       'MANAGER', SAL*1.20,
  5                                  SAL)
  6            REVISED_SALARY
  7  FROM      emp;
JOB              SAL REVISED_SALARY
--------- ---------- --------------
PRESIDENT        5000           5000
MANAGER          2850           3420
MANAGER          2450           2940
...
14 rows selected.
```



DONNÉES ISSUES DE PLUSIEURS TABLES

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.



LES TYPES DE JOINTURES

Il existe quatre types de jointures :

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

La condition de jointure doit être réalisée dans les clauses **JOIN** ou **WHERE.**

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 *

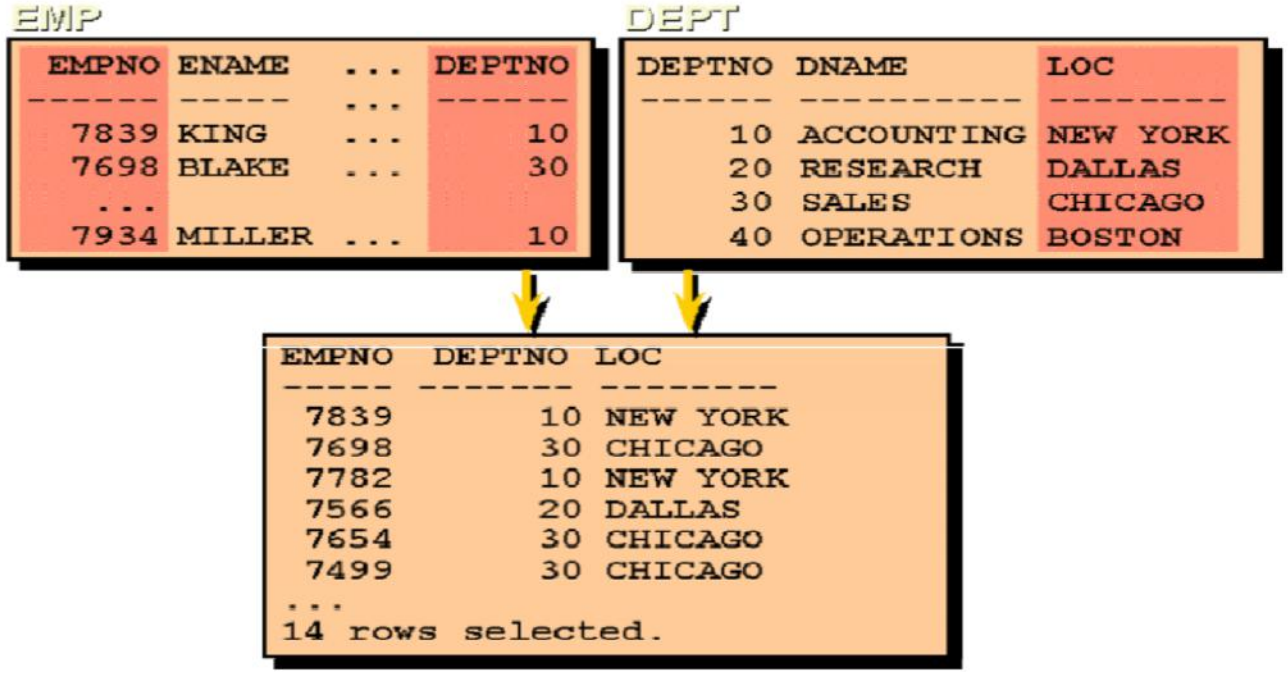
FROM emp,dept

WHERE emp.deptno=dept.deptno;

SELECT *

FROM emp

JOIN dept **ON** emp.deptno=dept.deptno;



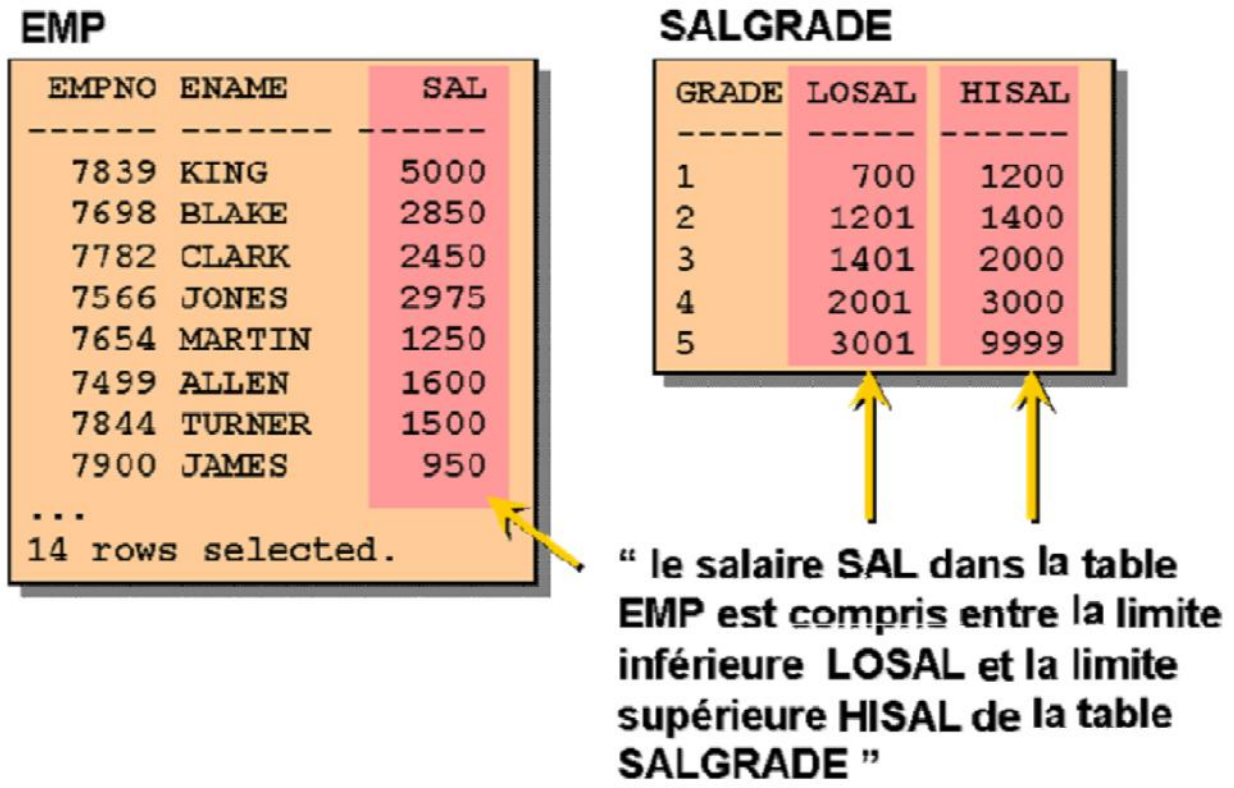
NON ÉQUI-JOINTURE

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

SELECT *

FROM emp,salgrade

WHERE sal BETWEEN losal and hisal;



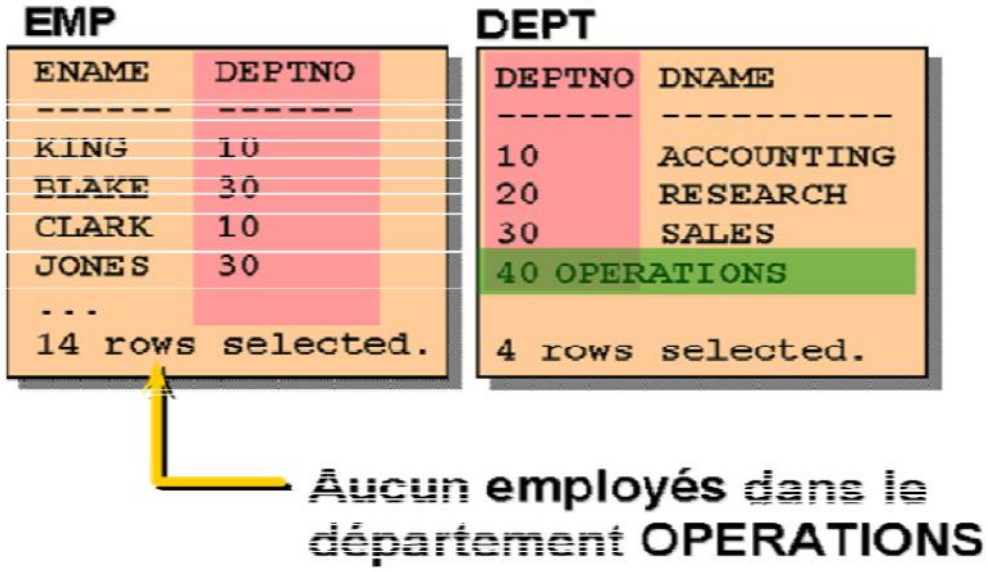
JOINTURE EXTERNE

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

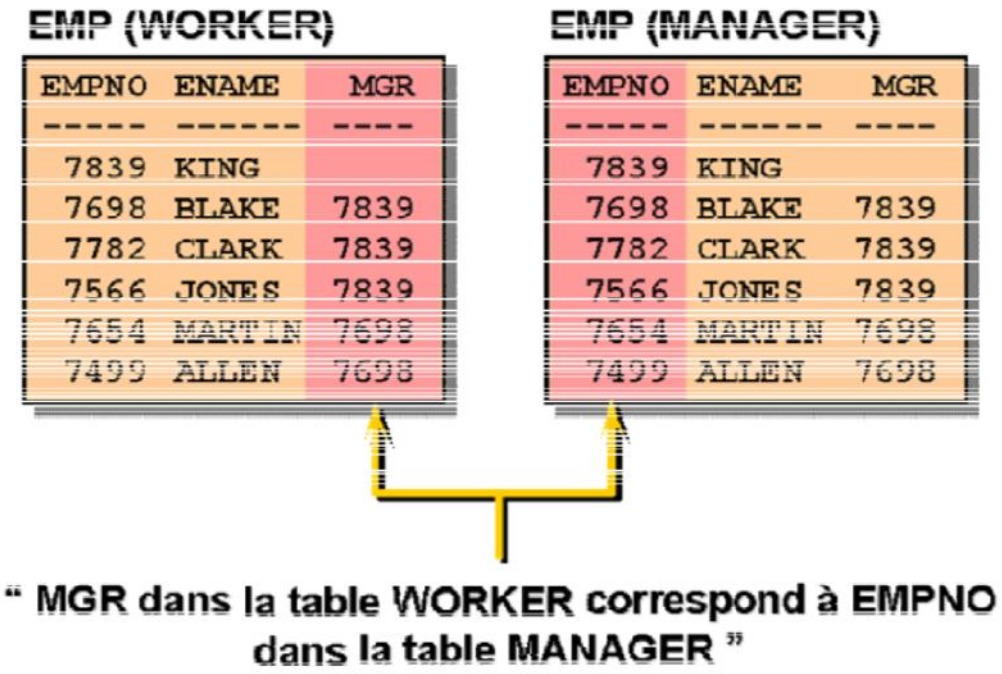
Cet requête affiche tous les enregistrements de la table 1 même si ils ne respectent pas la condition de jointure



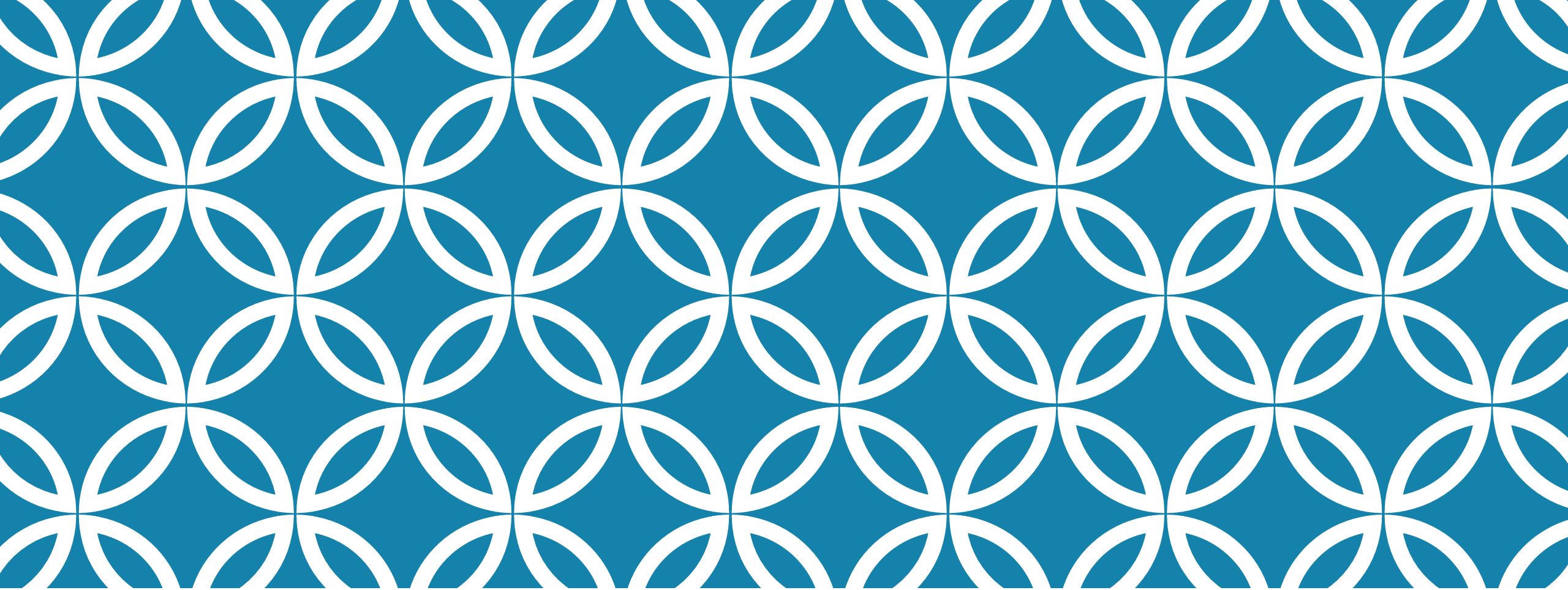
RELIER UNE TABLE À ELLE-MÊME AVEC AUTO-JOINTURE

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

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*).




```
SELECT *
FROM emp w,emp m
WHERE  w.mgr=m.empno;
```



LES FONCTIONS DE GROUPE

LES TYPES DE FONCTIONS DE GROUPE

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

Fonction	Résultat
SUM(n)	Retourne la somme de toutes les valeurs *n*
MIN(expr)	Retourne la plus petite valeur *expr*
MAX(expr)	Retourne la plus grande valeur *expr*
COUNT(expr)	Retourne le nombre d'enregistrements contenus dans *expr*.
AVG(n)	Retourne la moyenne des valeurs *n*

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.

UTILISATION DES FONCTIONS DE GROUPE

Affichage des lignes 0 - 0 (total de 1, Traitement en 0.0004 secondes.)

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

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans c

+ Options

AVG(sal)	MAX(sal)	MIN(sal)	SUM(sal)
1400.0000	1600	1250	5600

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 [*colonne1*,] *fonction_groupe*(*colonne2*)
FROM *table*
[**WHERE** *condition(s)*]
[**GROUP BY** *colonne1*]
[**ORDER BY** *colonnne2*] ;

deptno	AVG(sal)	MAX(sal)	MIN(sal)	SUM(sal)
NULL	1280.0000	1280	1280	1280
10	2916.6667	5000	1300	8750
20	2175.0000	3000	800	10875
30	1566.6667	2850	950	9400

LA CLAUSE GROUP BY

Quelques règles :

La clause WHERE peut être utilisée pour pré-exclure des enregistrements avant la division en groupes.

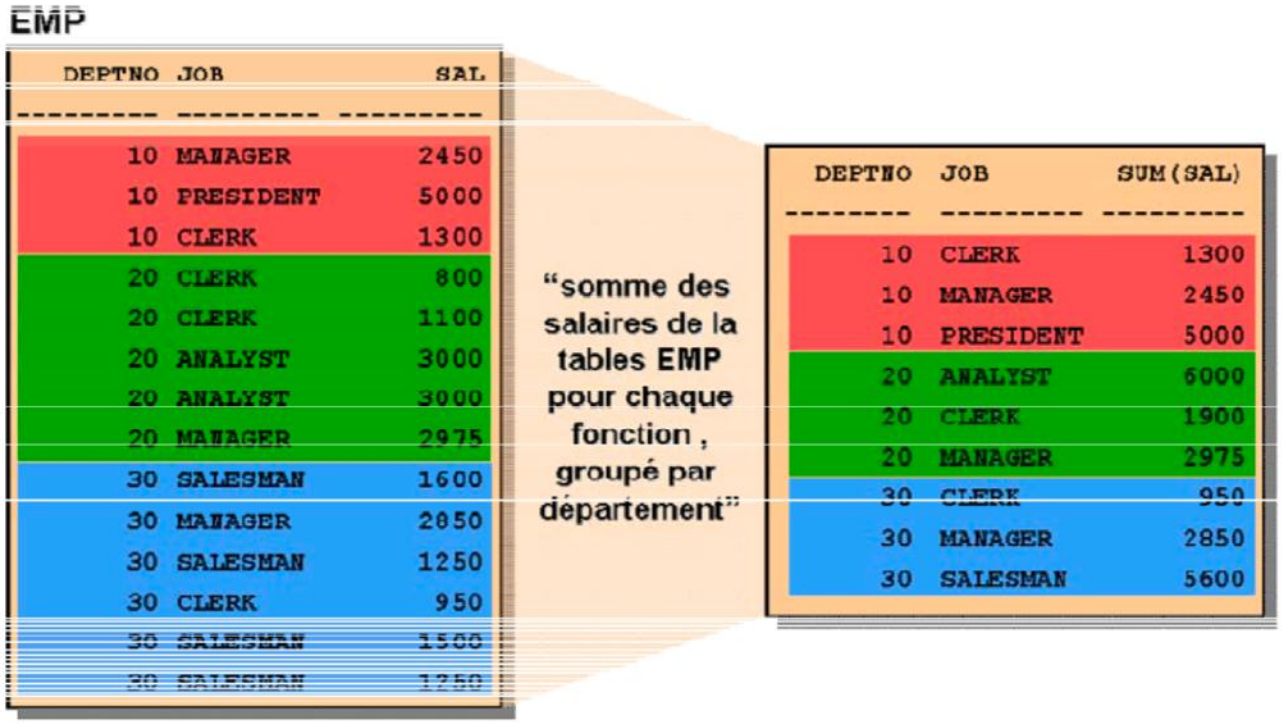
Les colonnes de la clause FROM qui ne sont pas inclues dans une fonction de groupe doivent être présentés dans la clause GROUP BY.

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.

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).



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 [*colonne1*,] *fonction_groupe*(*colonne2*)

FROM *table*

[**WHERE** *condition(s)*]

[**GROUP BY** *colonne1*]

[**HAVING** *condition de groupe*]

[**ORDER BY** *colonnne2*] ;

Affichage des lignes 0 - 1 (total de 2, Traitement en 0.0005 secondes.)

```
SELECT deptno,AVG(sal),MAX(sal),MIN(sal),SUM(sal) FROM emp GROUP BY deptno HAVING MIN(sal)>1000
```

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette ta

+ Options

deptno	AVG(sal)	MAX(sal)	MIN(sal)	SUM(sal)
NULL	1280.0000	1280	1280	1280
10	2916.6667	5000	1300	8750

REQUÊTES IMBRIQUÉES

Une sous-requête est une clause SELECT imbriquée dans une clause d'un autre ordre SQL.

Une sous-requête peut être utile lorsqu'il faut sélectionner des enregistrements en utilisant une condition qui dépend d'une valeur inconnue d'une autre colonne.

Exemple :

L'objectif est d'écrire une requête qui identifie tous les employés qui touchent un salaire plus grand que celui de l'employé Jones, mais la valeur du salaire de cet employé n'est pas connu.

Dans ce cas, il faut faire appel à une sous-requête qui va retourner le salaire de Jones à la requête principale.

Pour combiner deux requêtes, il suffira de placer une requête à l'intérieur d'une autre.

La requête à l'intérieure (ou la sous-requête) retourne une valeur qui est utilisée par la requête extérieure (ou requête principale).

L'utilisation d'une sous-requête est équivalente à l'utilisation de deux requêtes séquentielles.

Le résultat de la première requête est la valeur utilisée dans la seconde requête.

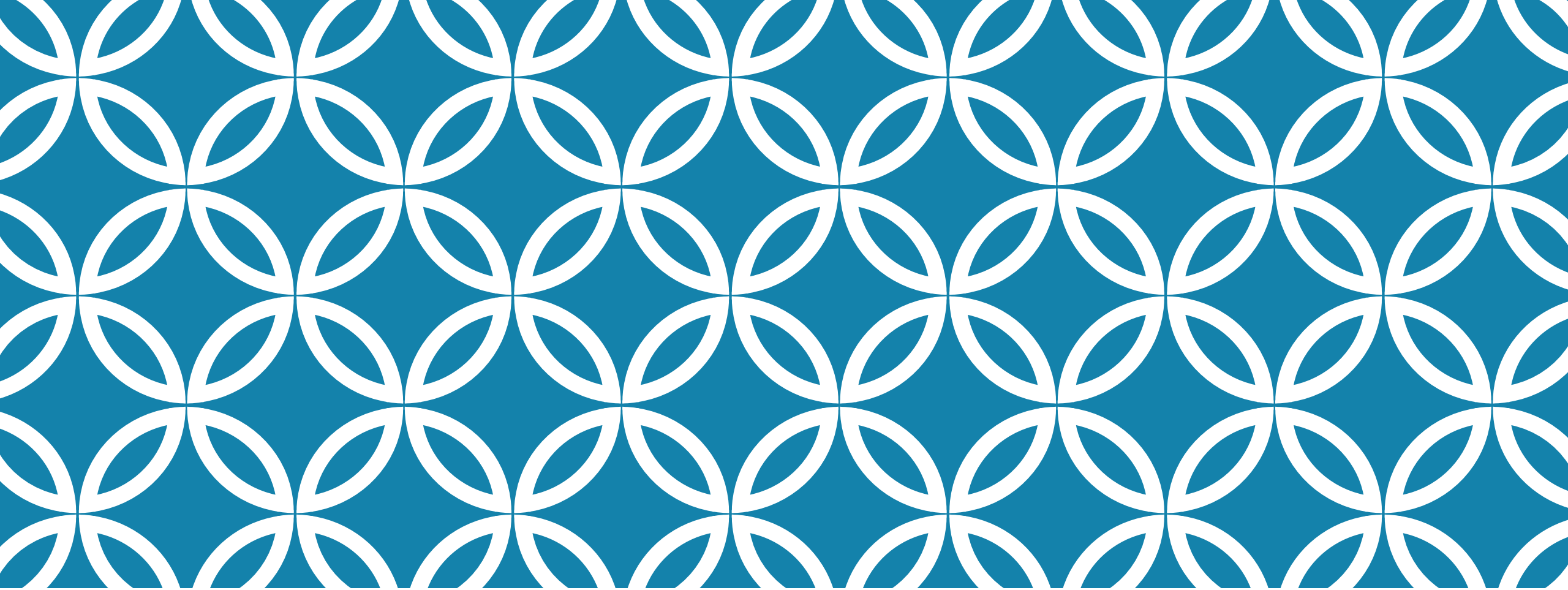
REQUÊTES IMBRIQUÉES

```
SELECT ename,sal from emp where sal>(SELECT sal from emp where ename="JONES")
```

+ Options

ename	sal
SCOTT	3000
KING	5000
FORD	3000

Règles de conduite :
Une sous-requête doit être mise entre parenthèses.
Une sous-requête doit être placée du coté 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.
Si une sous-requête retourne plusieurs valeurs, il faut utiliser les mots clés IN, ALL ou ANY



DML

MANIPULER LES DONNÉES

Les actions de modification, de rajout, de suppression des données dans une base de données exécutent un ordre SQL de type DML (Data Manipulation Language).

Il y a 3 ordres DML :

- INSERT : Pour insérer des données dans une table
- UPDATE : Pour modifier des données dans une table
- DELETE : Pour supprimer des données dans une table

INSERT

INSERT INTO nom_de_table

[(col1[, col2] ...)]

VALUES (value1 [, value2 ...]);

```
INSERT INTO EMP (empno, ename, hiredate)  VALUES
(4242, 'DUPOND', '21/08/01');
```

COPIER DES LIGNES D'UNE AUTRE TABLE

Il est possible grâce à l'ordre INSERT de copier les valeurs d'une table dans une autre sans avoir à en saisir les données.

Pour cela il suffit d'utiliser une requête SQL pour aller chercher les valeurs automatiquement.

```
INSERT INTO managers (id, name, salary, hiredate)
SELECT empno, ename, sal, hiredate
FROM emp
WHERE job='MANAGER';
```

Attention : La clause VALUES ne doit pas être utilisée dans ce type de requête. De plus le nombre de colonne passée dans la clause INTO doit correspondre au nombre de colonnes sélectionnées dans la requête SELECT.

UPDATE

Il est possible grâce à l'ordre UPDATE de modifier des valeurs de colonnes dans les tables.

Voici la syntaxe de l'ordre UPDATE :

UPDATE nom_table

SET col1=value1 [, col2=value2]

[**WHERE** condition]

```
UPDATE emp
SET comm=1000
WHERE ename='DUPOND'
```

DELETE

Pour effacer une ou plusieurs lignes d'une table il suffira d'utiliser la commande DELETE dont voici la syntaxe :

DELETE FROM table

WHERE condition;

```
DELETE FROM emp WHERE ename='DUPOND';
```