

# **COURS INGÉNIERIE DES BASES DE DONNÉES**

**Auditoire : 2<sup>ème</sup> année LSI ADBD - MM**

**Responsables du cours :**

Mme. Inès ZOUARI

[ines.zouari@isims.usf.tn](mailto:ines.zouari@isims.usf.tn)

Mme. Mariem MAHFOUDH

[mariemmahfoudh@gmail.com](mailto:mariemmahfoudh@gmail.com)

**Année universitaire : 2023-2024**

# Objectifs du cours

---

- ▶ Approfondir les connaissances sur les bases de données et le langage SQL
  - ▶ Réussir à structurer et résoudre des requêtes avancées sous Oracle
  - ▶ Maîtriser la définition et l'utilisation de vues
  - ▶ Se familiariser avec divers fonctions Oracle
- ▶ Apprendre à assurer la sécurité des bases de données à travers la gestion des utilisateurs et leurs droits
- ▶ Connaître les bases du langage PL/SQL

# Plan du cours

---

- ▶ Chapitre 1 : Le langage de requêtes SQL : SELECT
- ▶ Chapitre 2 : Les vues et les séquences
- ▶ Chapitre 3 : Le Langage de contrôle de données
- ▶ Chapitre 4 : Le langage PL/SQL

# Rappel : SQL (Structured Query Language)

---

- ▶ Langage de requêtes structuré
  - ▶ Créé en 1974, normalisé depuis 1986, SQL sert à exploiter les **bases de données relationnelles**
  - ▶ Il est composé de quatre parties :
    - ▶ Le Langage de Définition de Données (**LDD**) pour créer et supprimer des objets dans la BD (tables, contraintes d'intégrité, vues, etc.).
      - ➔ utilise les commandes : **CREATE, DROP, ALTER**
    - ▶ Le Langage de Manipulation de Données (**LMD**) pour la recherche, l'insertion, la mise à jour et la suppression de données
      - ➔ utilise les commandes : **INSERT, UPDATE, DELETE, SELECT**
    - ▶ Le Langage de Contrôle de Données (**LCD**) pour gérer les droits sur les objets de la base (création des utilisateurs et affectation de leurs droits)
      - ➔ utilise les commandes : **GRANT, REVOKE**
    - ▶ Le Langage de Contrôle de Transaction (**LCT**) pour la gestion des transactions : validation ou annulation de modifications de données dans la BD
      - ➔ utilise les commandes : **COMMIT, ROLLBACK**
- 





# **Chapitre I :**

## **Langage de requêtes SQL :**

### **SELECT**

# Plan du chapitre

---

1. Syntaxe générale du SELECT

2. Projection

3. Sélection

4. Fonctions spécifiques

5. Tri

6. Agrégation

7. Partitionnement

8. Requêtes imbriquées (simples, corrélées)

9. Produit cartésien

10. Jointure

11. Opérateurs ensemblistes

*Différentes utilisations du  
« SELECT »*



# I. Syntaxe générale du SELECT

---

```
SELECT [ { DISTINCT | UNIQUE } | ALL ] nomCol1 [ [AS] alias1 ], nomCol2 , ...  
FROM nomTable1 [aliasTable1] [, nomTable2 [aliasTable2] ]...  
[ WHERE condition ]  
[ GROUP BY ... ]  
[ HAVING condition ]  
[ { UNION | UNION ALL | INTERSECT | MINUS } ( sousRequête ) ]  
[ ORDER BY ... ] ;
```


- ▶ **UNIQUE** et **DISTINCT** éliminent les doublons
- ▶ **ALL** prend en compte les doublons
- ▶ « Select \* from ... » permet de sélectionner toutes les colonnes
- ▶ Les alias permettent de renommer des colonnes à l'affichage ou les tables dans la requête

## 2. Projection


**SELECT [DISTINCT | UNIQUE] liste\_colonnes FROM nomTable ;**

➔ La projection appliquée à une relation  $R$ , elle définit une relation restreinte à un sous-ensemble des attributs de  $R$ , en extrayant les valeurs des attributs spécifiés

**Exemples :** Soit la relation **Etudiant** (num, nom, prenom, age, ville, CodePostal)

 Afficher toute la table Etudiant  
**SELECT \* FROM Etudiant ;**

num	nom	prenom	age	ville	CodePostal
501	Tounsi	Salah	22	Sfax	3000
502	Bedoui	Islem	21	Gabes	6000
503	Selmi	Ahmed	21	Tunis	1001

 Donner les noms, les prénoms et les âges de tous les étudiants  
**SELECT nom, prenom, age FROM Etudiant ;**

 Donner les numéros des étudiants dans une colonne nommée Numéro  
**SELECT num AS Numéro FROM Etudiant ;**



### 3. Sélection

```
SELECT * FROM nomTable WHERE condition ;
```

➔ La sélection est une opération sur une relation qui retourne une relation de même schéma mais avec uniquement les tuples qui vérifient une **condition** spécifiée en argument

➔ La **condition** peut être formée sur des noms de colonnes ou des constantes avec :

▶ + , - , \* , / , = , <> , != , > , < , >= , <=

▶ **AND, OR, NOT**

▶ **BETWEEN** *Val1* **and** *Val2* : permet de tester si le contenu d'une colonne est compris entre  $Val_1$  et  $Val_2$

▶ **IS NULL** : permet de tester si le contenu d'une colonne est une valeur nulle (indéfinie)

▶ **IN** (*liste de valeurs*) : permet de tester si le contenu d'une colonne coïncide avec l'une des valeurs de la liste.

### 3. Sélection (suite)

---

- ▶ *Op. comparaison* **ALL** *sous\_requête* : Vérifier si une valeur est =, !=, >=, < ou <= pour tous les résultats retournés par une sous-requête.
- ▶ *Op. comparaison* **ANY** *sous\_requête* : Vérifier si une valeur est =, !=, >, >=, < ou <= pour au moins une des valeurs de la sous-requête.
- ▶ **LIKE** *chaîne générique* : permet de tester si le contenu d'une colonne ressemble à une chaîne de caractères obtenues à partir de la chaîne générique. La chaîne générique est une chaîne de caractères qui contient l'un des caractères suivants :
  - ▶ **%** : remplace une autre chaîne de caractères qui peut être même une chaîne vide.
  - ▶ **\_** : remplace un seul caractère.
- ▶ Tous les opérateurs spécifiques peuvent être mis sous forme négative en les faisant précéder de l'opérateur de négation **NOT** : **NOT IN, NOT BETWEEN, NOT LIKE, IS NOT NULL.**



### 3. Sélection (suite)

#### Exemples

Etudiant (Num, Nom, Prenom, Age, Ville, CodePostal)

num	nom	prenom	age	ville	Code postal
490	Salem	Sami	19	BebSaad	1029
491	Adli	Fatma	20	Kef	7100
492	Ali	Alaa	19	Mahdia	5100
493	Atia	Ahmed	21	Tunis	1000
494	Turki	Nour	23	SakietEz	3021

Quels sont tous les étudiants âgés de 20 ans ou plus ?

SELECT \* FROM Etudiant WHERE Age >= 20 ;

Quels sont tous les étudiants âgés de 19 à 23 ans ?

SELECT \* FROM Etudiant WHERE Age IN (19, 20, 21, 22, 23) ;

SELECT \* FROM Etudiant WHERE Age BETWEEN 19 AND 23 ;

Quels sont tous les étudiants habitant à TUNIS ?

SELECT \* FROM Etudiant WHERE CodePostal LIKE '10%' ;

Quels sont tous les étudiants dont la ville est inconnue/connue ?

SELECT \* FROM Etudiant WHERE Ville IS NULL ;

SELECT \* FROM Etudiant WHERE Ville IS NOT NULL ;

### 3. Sélection (suite)

**Emp** (id\_emp, nom, sal, indice, adresse, #idService)

**Service** (idService, nomService)

#### Exemple (Projection + Sélection)

→ Afficher les noms et les salaires des employés n'appartenant pas aux services : 10, 40, 60

```
SELECT nom AS "Nom Employé", sal AS Salaire  
FROM Emp  
WHERE idService NOT IN (10,40,60) ;
```

Emp

	id_emp	nom	sal	indice	adresse	idService
	100	Ali	1500	...	...	10
→	101	Salah	1600	...	...	20
→	102	Med	1800	...	...	20
→	103	Ahmed	2200	...	...	30
	104	Alia	2000	...	...	40
	105	Hatem	2500	...	...	40
→	106	fatma	1900	...	...	50

#### Résultat

Nom Employé	Salaire
Salah	1600
Med	1800
Ahmed	2200
fatma	1900

## 4. Fonctions spécifiques – arithmétiques - date

Dans les clauses **SELECT** et **WHERE**, on peut utiliser des fonctions arithmétiques telles que :

- ▶ **ABS(n)** : permet de calculer la valeur absolue de n.
- ▶ **CEIL(n)** : permet d'avoir le plus petit entier supérieur ou égal à n.
- ▶ **FLOOR(n)** : permet d'avoir la partie entière de n.
- ▶ **MOD(m, n)** : permet d'avoir le reste de la division entière de m par n.
- ▶ **ROUND(m, n)** : arrondit la valeur de m à n décimal.
- ▶ **POWER(m, n)** : permet d'avoir m puissance n.
- ▶ **SIGN(n)** : donne -1 si n < 0, donne 0 si n = 0 et donne 1 si n > 0.
- ▶ **SQRT(n)** : permet d'avoir  $\sqrt{n}$ .
- ▶ **TRUNC(m, n)** : permet de tronquer la valeur après n décimales. Si n est négatif, la valeur de m est tronquée avant le point décimal.

Des fonctions de manipulation de date, telles que :

- ▶ **ADD\_MONTHS(d, n)** : permet d'ajouter n mois à la date d ; n est un entier.
- ▶ **GREATEST(d1, d2)** : permet d'avoir la date la plus récente parmi d1 et d2 ≠ **LEAST**.
- ▶ **MONTHS\_BETWEEN(d1, d2)** : permet d'avoir le nombre de mois qui se trouvent entre la date d1 et la date d2.
- ▶ **LAST\_DAY(d)** : permet de retourner la date du dernier jour du mois de la date d.
- ▶ **SYSDATE** : donne la date et l'heure système.

## 4. Fonctions spécifiques – chaînes de caractères

---

- ▶ **RTRIM(ch)** : supprime l'espace à la fin de la chaîne  $\neq$  **LTRIM(ch)**
- ▶ **RPAD(ch1, n [, ch2])** : ajoute ch2 à la fin de ch1 « autant de fois » jusqu'à atteindre la taille n  $\neq$  **LPAD(ch1, n, ch2)**  
**RPAD(ch, n)** : ajoute des espaces à la fin de ch jusqu'à atteindre la taille n  $\neq$  **LPAD(ch, n)**
- ▶ **INITCAP(ch)** : met en majuscule la première lettre de chaque mot de la chaîne
- ▶ **INSTR(ch1, ch2 [, n [, m]])** : cherche la position de la sous-chaîne ch2 dans la chaîne ch1 à partir de la position n et si l'on souhaite , à partir de la **m**<sup>ième</sup> occurrence de **ch2** dans **ch1**. **n, m** : sont optionnels et par défaut = 1
- ▶ **LENGTH(ch)** : renvoie la longueur d'une chaîne
- ▶ **LOWER(ch)** : transforme la chaîne **ch** en minuscule  $\neq$  **UPPER(ch)**
- ▶ **SUBSTR(ch, m, n)** : permet d'extraire une sous-chaîne de **ch** commençant à partir du caractère de position **m** et de longueur **n**
- ▶ **ch1 || ch2** : concatène les deux chaînes
- ▶ **TRANSLATE(ch, ch1, ch2)** : permet de transformer dans la chaîne ch les caractères de **ch1** par ceux de ch2.
- ▶ **Replace(chaine, ch1 [, ch2])** : remplace une chaîne par une autre dans une colonne. Si on ne met pas **ch2**, **ch1** va être remplacée par un vide



## 4. Fonctions spécifiques - conversion

---

- ▶ **TO\_NUMBER(ch[,format])** : convertit une chaîne de caractères contenant des chiffres en valeur de type NUMBER.
- ▶ **NVL(arg1, arg2)** : La fonction NVL retourne arg2 si arg1 est null, sinon la fonction retourne arg1  
Les paramètres utilisés par NVL peuvent être de tout type de données
- ▶ **DECODE(expr, val\_1, res\_1 [, val\_2, res\_2 ...], def)** : Cette fonction permet de choisir une valeur parmi une liste d'expressions, en fonction de la valeur prise par une expression servant de critère de sélection. Le résultat récupéré est :
  - ▶ **res\_1** : si l'expression **expr** a la valeur val\_1
  - ▶ **res\_2** : si l'expression **expr** a la valeur val\_2
  - ▶ **def** (la valeur par défaut) : si l'expression **expr** n'est égale à aucune valeurs : val\_1, val\_2, ...,.
- ▶ **TO\_DATE (chaîne[,format\_date])** : convertit une chaîne de caractères (1<sup>er</sup> paramètre) ayant le format (2<sup>ème</sup> paramètre) à une valeur de type Date.
- ▶ **TO\_CHAR (date, [,format\_date]) / TO\_CHAR (nombre[,format\_nombre])** : convertit une date ou une valeur numérique en chaîne de caractères.



## 4. Fonctions spécifiques - conversion

---

Parmi les formats de dates on cite :

- ▶ **YYYY**      Année
  - ▶ **YYY**      3 derniers chiffres de l'année
  - ▶ **YY**      2 derniers chiffres de l'année
  - ▶ **Y**      Dernier chiffre de l'année
  - ▶ **Q**      Numéro de trimestre de l'année (1 à 4)
  - ▶ **WW**      Numéro de semaine de l'année (1 à 52)
  - ▶ **W**      Numéro de semaine dans le mois
  - ▶ **MM**      Numéro du mois (01 à 12)
  - ▶ **DDD**      Numéro de jour dans l'année (1 à 366)
  - ▶ **DD**      Numéro du jour dans le mois (1 à 31)
  - ▶ **D**      Numéro de jour dans la semaine (1 à 7)
  - ▶ **YEAR**      Année en toute lettre
  - ▶ **MON**      Nom du mois abrégé
  - ▶ **DAY**      Nom du jour de la semaine sur 9 caractères
  - ▶ **DY**      Nom du jour de la semaine abrégé en 3 lettres
- 





## 4. Fonctions spécifiques - Exemples

Select **TRUNC** (121.371,1) from Dual; → 121.3

Select **TRUNC** (121.371,-1) from Dual; → 120

Select **ROUND**(15.193,1) from Dual; → 15.2

Select **ROUND**(15.193,-1) from Dual; → 20

Select **ROUND**(12.193,-1) from Dual; → 10

select **CEIL**(3.5) from Dual; → 4

Select **FLOOR**(3.5) from Dual; → 3

Select **TRANSLATE** ('1tech23','123','456') from Dual → '4tech56'

Select **REPLACE**('1212tech','12') from Dual ; → 'tech' //Select **REPLACE**('12tech','12','ab') from Dual; → 'abtech'

SELECT **INSTR** ('Mednine','e',3,1) FROM Dual ; → 7

Select **SUBSTR**('Janvier',2,3) FROM Dual ; → 'anv'

Select **SYSDTATE** from Dual; → 20/09/2023 // Select **ADD\_MONTHS**(sysdate, 3) from dual; → 20/12/2023

Select **MONTHS\_BETWEEN** (sysdate,'01/06/2023') from Dual; → 3,61290323

SELECT **TO\_DATE**('10-12-22','MM-DD-YY') FROM Dual; → 12/10/2022

Select **TO\_CHAR** (120.73,'999.9') From Dual; → '120.7' // Select **TO\_CHAR** (21,'0099') From Dual; → ' 0021'

Select **TO\_CHAR** (sysdate,'yyyy') FROM Dual; → ' 2023'

Select **TO\_CHAR** (sysdate,'Month DD,YYYY') FROM Dual; → ' September 20, 2023'



## 5.Tri de résultats

### Syntaxe :

**ORDER BY** { *expression1* | *position1* | *alias1* } [ASC | DESC] [, {*expression2* | *position2* | *alias2*} [ASC | DESC] ...

- ▶ *expression* : nom de colonne, fonction, constante, calcul.
- ▶ *position* : entier qui désigne l'expression (au lieu de la nommer) dans son ordre d'apparition dans la clause SELECT.
- ▶ ASC ou DESC : tri ascendant ou descendant (par défaut ASC).

**Exemple :** **Etudiant** (Num, Nom, Prenom, Age, Ville, CodePostal)

*Trier la table Etudiant par âge décroissant et par nom croissant*

**SELECT \* FROM Etudiant ORDER BY age DESC, nom ASC ;**

num	nom	prenom	age	ville	CodePostal
...	Abbes	...	22	...	...
...	Ben Amor	...	22	...	...
...	Tounsi	...	22	...	...
...	Bedoui	...	21	....	...
...	Selmi	...	21	...	...

## 6. Agrégation des résultats

---

Il est possible d'appliquer des fonctions d'agrégation au résultat d'une sélection

### Syntaxe :

**SELECT  $f$  ( [ALL | DISTINCT] expression) FROM ...**

où $f$ peut être	COUNT	nombre de tuples
	SUM	somme des valeurs d'une colonne
	AVG	moyenne des valeurs d'une colonne
	MAX	maximum des valeurs d'une colonne
	MIN	minimum des valeurs d'une colonne
	STDDEV	permet d'avoir l'écart type
	VARIANCE	permet d'avoir la variance

Pour COUNT, on peut aussi utiliser COUNT(\*)

Seul COUNT prend en compte les valeurs à NULL.



## 6. Agrégation des résultats

Résultats (de Salah)

<b>Matière</b>	<b>Coef</b>	<b>Note</b>
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

**Quelle est la meilleure note de Salah ?**

`SELECT MAX(Note) FROM Résultats ;` → 15

**Quelle est sa plus mauvaise note ?**

`SELECT MIN(Note) FROM Résultats ;` → 9

**Quelle est la somme pondérée des notes ?**

`SELECT SUM(Note*Coef) FROM Résultats ;` → 193

**Quelle est la moyenne de Salah ?**

`SELECT SUM(Note*Coef) / SUM(Coef) FROM Résultats ;` → 12,06

**Dans combien de matières Salah a-t-il eu plus de 12 ?**

`SELECT COUNT(*) FROM Résultats WHERE Note > 12 ;` → 2



# 7. Partitionnement des résultats

## Syntaxe

```
SELECT col1 [, col2...], [fonction1 Groupe(...)] [, fonction2 Groupe(...)]  
FROM nomTable  
[ WHERE condition ]  
GROUP BY col1 [, col2]...)  
[ HAVING condition ]  
[ ORDER BY... ] ;
```

	col1	col2	col3	col4
Groupement 1	...	val1	...	...
	...	val1	...	...
Groupement 2	...	val2	...	...
	...	val2	...	...
	...	val2	...	...
Groupement 3	...	val3	...	...

Dans l'ordre, on effectue :

- la sélection SELECT
- le partitionnement GROUP BY
- on retient les partitions intéressantes HAVING
- on trie avec ORDER BY

☞ la clause **WHERE** s'applique donc à la totalité de la table

☞ la clause **HAVING** permet de poser des conditions sur chaque groupement

## 7. Partitionnement des résultats

### Exemple :

Résultats (de Salah)

<i>Matière</i>	<i>Coef</i>	<i>Note</i>
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

**Quelle est la note moyenne pour chaque coefficient ?**

```
SELECT coef, Avg(note) as Moyenne  
FROM Résultats  
GROUP BY coef;
```

<i>Coef</i>	<i>Moyenne</i>
1	11
2	12
3	10.5
4	15

**Quels sont les coefficients auxquels participe une seule matière ?**

```
SELECT coef  
FROM Résultats  
GROUP BY coef  
HAVING count(*)=1;
```

<i>Coef</i>
4



## 8. Requêtes imbriquées

---

- ▶ On parle de requête imbriquée lorsqu'une sous requête SELECT apparaît dans la clause WHERE ou HAVING de la requête principale SELECT
- ▶ Il existe 2 types de requêtes imbriquées :
  - ▶ **Requêtes imbriquées simples**
    - ▶ la requête interne **est indépendante** de la requête externe càd : la req. interne est complètement évaluée **avant** la req. externe qui utilisera ensuite le résultat
    - ▶ **N.B.** La sous requête interne peut retourner une ou plusieurs valeurs
  - ▶ **Requêtes imbriquées corrélées (synchronisées )**
    - ▶ La sous requête interne dépend de la requête externe
    - ▶ La sous-requêtes corrélée fait référence à **la ligne actuelle** de sa requête externe

## 8. Requêtes imbriquées

### Requêtes imbriquées simples

**Exemple 1 :** **PRODUIT** (cod\_pd, designation, categ, prixU, qtité\_stock)

→ Trouver la désignation des produits dont le prix unitaire est égal à celui du produit 'chaise'

```
SELECT designation FROM produit
WHERE prixU = ( SELECT prixU FROM produit WHERE designation = 'chaise' );
```

150

**Produit**

cod_pd	designation	categ	prixU	Qtite_stock
99	table	meuble	350	20
100	Armoire	meuble	1200	5
101	bureau	bureautique	520	15
102	chaise	bureautique	150	120
103	lustre	décor	180	12
104	tableau	décor	150	13

**Résultat**

designation
chaise
tableau



## 8. Requêtes imbriquées

### Requêtes imbriquées simples

**Exemple 2:** **PRODUIT** (cod\_pd, designation, categ, prixU, qtité\_stock)

→ Trouver la désignation des produits dont le prix est supérieur à tous les produits de la catégorie «bureautique »

```
SELECT designation FROM Produit
WHERE prixU > All ( SELECT prixU FROM produit WHERE categ = 'bureautique' );
```

{ 520, 150 }

Produit

Cod_pd	designation	categ	PrixU	Qtite_stock
99	table	meuble	350	20
100	Armoire	meuble	1200	5
101	bureau	bureautique	520	15
102	chaise	bureautique	150	120
103	lustre	décor	180	12
104	tableau	décor	150	13

Résultat

designation  
-----  
Armoire

## 8. Requêtes imbriquées

### Requêtes imbriquées corrélées (synchronisées)

**Exemple 1 :** Emp (id\_emp, nom, sal, indice, adresse, #idService)

Afficher tout employé ayant un salaire supérieur au salaire moyen de son service

```
SELECT id_emp
FROM Emp EI
WHERE Sal > ( SELECT AVG(Sal) FROM Emp E2 WHERE EI.idService = E2.idService );
```

Résultat

id\_emp

101

103

105

EI

id_emp	nom	sal	indice	adr	EI.idService
100	Ali	1500	...	...	10
101	Salah	1600	...	...	10
102	Med	1800	...	...	20
103	Ahmed	2200	...	...	20
104	Alia	2000	...	...	20
105	Hatem	2500	...	...	30
106	fatma	1900	...	...	30

E2

id_emp	nom	sal	indice	adr	E2.idService
100	Ali	1500	...	...	10
101	Salah	1600	...	...	10
102	Med	1800	...	...	20
103	Ahmed	2200	...	...	20
104	Alia	2000	...	...	20
105	Hatem	2500	...	...	30
106	fatma	1900	...	...	30

## 8. Requêtes imbriquées

### Requêtes imbriquées corrélées avec EXISTS

- L'opérateur EXISTS s'utilise dans une clause WHERE pour savoir s'il y a présence ou non de lignes retournées par la sous-requête

```
SELECT nom_colonne1, ...  
FROM nomTable1 WHERE EXISTS ( SELECT nom_colonne2, ...  
                                FROM nomTable2  
                                WHERE condition );
```

- ☞ l'opérateur EXISTS retourne TRUE si la sous requête retourne au moins un résultat (une ligne) → condition pour que la requête externe s'exécute
- ☞ l'opérateur EXISTS retourne FALSE si la sous requête ne retourne aucune ligne

## 8. Requêtes imbriquées

### Requêtes imbriquées corrélées avec EXISTS

**Exemple :** EMPLOYE(idemp, nom, salaire, #idemp\_chef, #num\_service)

Quels sont les employés ayant au moins un employé sous leurs responsabilité ?

```
SELECT e1.idemp, e1.nom
FROM employe e1 WHERE EXISTS (SELECT e2.*
                              FROM employe e2
                              WHERE e2.idemp_chef = e1.idemp) ;
```

e1				
idemp	nom	salaire	Idemp_chef	NumServ
100	Ali	1500	103	10
101	Salah	1500	103	10
102	Med	1800	104	20
103	Ahmed	2200	105	10
104	Alia	2000	105	20
105	Hatem	2500	null	30

e2				
idemp	nom	salaire	Idemp_chef	NumServ
100	Ali	1500	103	10
101	Salah	1500	103	10
102	Med	1800	104	20
103	Ahmed	2200	105	10
104	Alia	2000	105	20
105	Hatem	2500	null	30

## 8. Requêtes imbriquées

### Requêtes imbriquées corrélées avec **NOT EXISTS**

---

☞ l'opérateur **NOT EXISTS** retourne TRUE si aucune ligne n'est retournée par la sous requête → condition pour que la requête externe s'exécute

```
SELECT nom_colonne1, ...  
FROM nomTable1 WHERE NOT EXISTS ( SELECT nom_colonne2, ...  
                                     FROM nomTable2  
                                     WHERE condition );
```

## 8. Requêtes imbriquées

**Emp** (id\_emp, nom, sal, indice, adresse, #idService)  
**Service** (idService, nomService)

### Requêtes imbriquées corrélées avec **NOT EXISTS**

**Exemple :** Quels sont les services n'ayant pas d'employés ?

```
SELECT s.*  
FROM service s WHERE NOT EXISTS (SELECT e.num_service  
FROM employe e  
WHERE e.num_service = s.num_service) ;
```

service	
s.num_Service	nom_service
10	RH
20	INFORMATIQUE
30	MARKETING
40	COMPTABILITE

employe				
idemp	nom	salaire	.....	e.num_service
100	Ali	1500		10
101	Salah	1600		10
102	Med	1800		20
103	Ahmed	2200		20
104	Alia	2000		20
105	Hatem	2500		30
106	fatma	1900		30

Résultat	Num_Service	Nom_Service
▶	40	COMPTABILITE

## 9. Produit cartésien

---

**SELECT \* FROM** *nomTable<sub>1</sub>*, ..., *nomTable<sub>n</sub>* ;


- ▶ Le produit cartésien est une opération portant sur deux relations R1 et R2 qui n'ont pas nécessairement le même schéma
- ▶ Il consiste à construire une relation R3 ayant pour schéma la concaténation des schémas de R1 et R2 et contenant toutes les concaténations possibles des tuples des deux relations.



## 9. Produit cartésien

**Exemple : Produit** (NumP, LibP, Coul, Poids)

**Client** (NumCI, NomCI, AdrCI)

 Lister tous les achats possibles des clients (*produits pouvant être achetés par tous les clients*)  
`SELECT * FROM Client, Produit ;`

NumCI	NomCI	AdrCI
CL01	Batam	Sfax
CL02	AMS	Sousse
CL03	AMS	Tunis

NumP	LibP	Coul	Poids
P001	Robinet	Gris	5
P002	Prise	Blanc	1.2

NumCI	NomCI	AdrCI	NumP	LibP	Coul	Poids
CL01	Batam	Sfax	P001	Robinet	Gris	5
CL01	Batam	Sfax	P002	Prise	Blanc	1.2
CL02	AMS	Sousse	P001	Robinet	Gris	5
CL02	AMS	Sousse	P002	Prise	Blanc	1.2
CL03	AMS	Tunis	P001	Robinet	Gris	5
CL03	AMS	Tunis	P002	Prise	Blanc	1.2



# I 0. Jointure

---

```
SELECT * FROM Table1 [Alias1], ..., Tablen [Aliasn] WHERE condition_jointure ;
```

## Autre Syntaxe :

```
SELECT * FROM table1 INNER JOIN table2 ON condition;
```

- ▶ Les jointures en SQL permettent d'associer plusieurs tables dans une même requête.
- ▶ Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace
- ▶ Les jointures consistent à associer des lignes de 2 tables T1 et T2 en associant l'égalité des valeurs d'une colonne de T1 par rapport à la valeur d'une colonne de T2.

# I 0. Jointure

- **Exemple :** Lister les noms des clients qui ont passé des commandes ?

**Client**

NCli	NomCl	AdrCl
CL01	Batam	Sfax
CL02	AMS	Sousse
CL03	BIAS	Monastir

**Commande**

NCmd	DatCmd	NCI
C001	10/12/2018	CL01
C002	13/02/2019	CL03
C003	03/09/2019	CL01

**Commande x Client**

NCmd	DatCmd	NCI	NCli	NomCl	AdrCl
C001	10/12/2018	CL01	CL01	Batam	Sfax
C001	10/12/2018	CL01	CL02	AMS	Sousse
C001	10/12/2018	CL01	CL03	BIAS	Monastir
C002	13/02/2019	CL03	CL01	Batam	Sfax
C002	13/02/2019	CL03	CL02	AMS	Sousse
C002	13/02/2019	CL03	CL03	BIAS	Monastir
C003	03/09/2019	CL01	CL01	Batam	Sfax
C003	03/09/2019	CL01	CL02	AMS	Sousse
C003	03/09/2019	CL01	CL03	BIAS	Monastir

**Commande** ⋈ **Client** [Commande.NCI=Client.NCli]

NCmd	DatCmd	NCI	NCli	NomCl	AdrCl
C001	10/12/2018	CL01	CL01	Batam	Sfax
C002	13/02/2019	CL03	CL03	BIAS	Monastir
C003	03/09/2019	CL01	CL01	Batam	Sfax

En appliquant le prédicat :  
Commande.NCI = Client.NCli

**Résultat**

NomCl
Batam
BIAS
Batam

# I 0. Jointure

---

## Exemple

***Produit** (prod, nomProd, fournisseur, pu)  
**DétailCommande** (#cmd, #prod, pu, qte, remise)*

***Quels sont les numéros de commande correspondant à l'achat d'un cahier ?***

```
SELECT DétailCommande.cmd  
FROM Produit, DétailCommande  
WHERE Produit.prod = DétailCommande.prod  
AND    nomProd LIKE '%cahier%';
```

***Même requête, mais avec des alias pour les noms de relation :***

```
SELECT cmd  
FROM Produit p, DétailCommande dc  
WHERE p.prod = dc.prod  
AND    nomProd LIKE '%cahier%';
```



# I 0. Jointure

---

## Jointure par requêtes imbriquées

Une jointure peut aussi être effectuée à l'aide d'une sous-requête.

```
SELECT cmd  
FROM DétailCommande  
WHERE prod IN (SELECT prod FROM Produit  
                WHERE nomProd LIKE '%cahier%');
```

← Sous-requête imbriquée



Dans ce cas, la sous-requête ne doit retourner qu'une colonne !

# I 0. Jointure

---

## Types de Jointure

- ▶ Jointure interne (INNER JOIN)
- ▶ Jointure externe :
  - ▶ LEFT JOIN (LEFT OUTER JOIN)
  - ▶ RIGHT JOIN (RIGHT OUTER JOIN)
  - ▶ FULL JOIN (FULL OUTER JOIN)
- ▶ Jointure croisée (CROSS JOIN)
- ▶ Auto-jointure (SELF JOIN)

# 10. Jointure

## Types de Jointure : INNER JOIN

- ▶ La commande **INNER JOIN** est aussi appelée EQUIJOIN
- ▶ La syntaxe suivante stipule qu'il faut sélectionner les enregistrements des tables : table1 et table2 lorsque la valeur de la colonne id de table1 est égale à la valeur de la colonne fk\_id de table2.

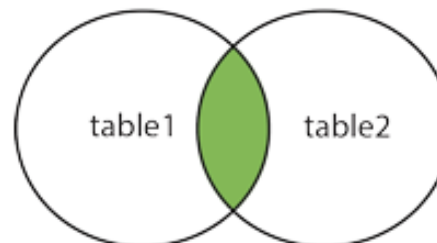
```
SELECT col1, col2, ..., coln  
FROM table1 [INNER] JOIN table2  
ON table1.id = table2.fk_id ;
```

ou

*Ancienne syntaxe de SQL*

```
SELECT col1, col2, ..., coln  
FROM table1, table2  
WHERE table1.id = table2.fk_id ;
```

INNER JOIN



# I 0. Jointure

## Types de Jointure : INNER JOIN

**COMPAGNIE** (ID\_COMP, NOM\_COMP, ADRESSE)

**PILOTE** ( NUM\_PIL, NOM\_PIL, NBHVOL, #ID\_COMPA )

Compagnie		
ID_COMP	NOM_COMP	ADRESSE
AF	AIR FRANCE	Paris
SING	SINGAPORE AL	Singapour
TA	TUNISAIR	Tunis

Pilote			
NUM_PIL	NOM_PIL	NBHVOL	ID_COMPA
104	Alain Lebeau	-	-
100	Pierre Lamothe	450	AF
101	Didier Linxe	800	AF
102	Ali Salah	1100	TA
103	Henri Alquié	930	AF

**Exemple :** Afficher les noms de pilotes de la compagnie 'AIR FRANCE'

```
SELECT NOM_PIL
FROM COMPAGNIE INNER JOIN PILOTE
ON ID_COMP = ID_COMPA
WHERE NOM_COMP = 'AIR FRANCE' ;
```



NOM_PIL
Pierre Lamothe
Didier Linxe
Henri Alquié

# I 0. Jointure

## Types de Jointure : LEFT JOIN

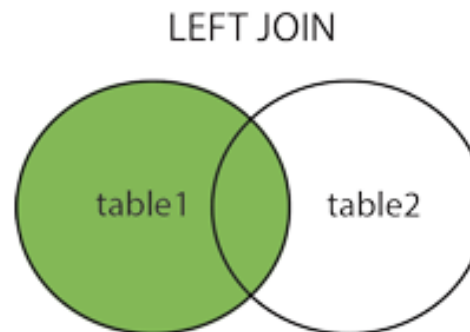
- ▶ Appelée aussi **LEFT OUTER JOIN** permet de lister tous les résultats de la table de gauche même s'il n'y a pas de correspondance dans la deuxième table.

### Syntaxe:

```
SELECT col1, col2, ..., coln
```

```
FROM table1 LEFT [OUTER] JOIN table2 ON table1.id = table2.fk_id;
```

### Schéma:





# I 0. Jointure

## Types de Jointure : LEFT JOIN

### Exemple :

Compagnie		
ID_COMP	NOM_COMP	ADRESSE
AF	AIR FRANCE	Paris
SING	SINGAPORE AL	Singapour
TA	TUNISAIR	Tunis

Pilote			
NUM_PIL	NOM_PIL	NBHVOL	ID_COMPA
104	Alain Lebeau	-	-
100	Pierre Lamothe	450	AF
101	Didier Linxe	800	AF
102	Ali Salah	1100	TA
103	Henri Alquié	930	AF

➔ Afficher la liste des compagnies et leurs pilotes incluant les compagnies n'ayant pas de pilotes

```
SELECT NOM_COMP, NOM_PIL
FROM COMPAGNIE LEFT JOIN PILOTE
ON ID_COMP = ID_COMPA ;
```



NOM_COMP	NOM_PIL
AIR FRANCE	Pierre Lamothe
AIR FRANCE	Didier Linxe
TUNISAIR	Ali Salah
AIR FRANCE	Henri Alquié
SINGAPORE AL	-

# I 0. Jointure

## Types de Jointure : RIGHT JOIN

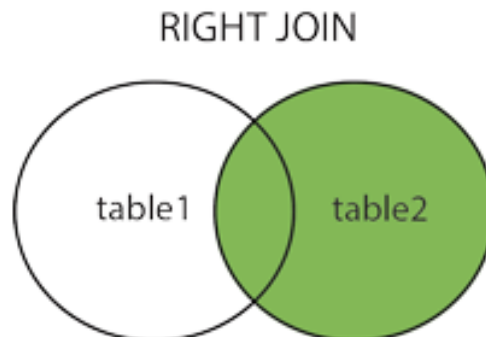
- ▶ La commande **RIGHT JOIN** (ou **RIGHT OUTER JOIN**) permet de retourner tous les enregistrements de la table de droite même s'il n'y a pas de correspondance avec la table de gauche. S'il y a un enregistrement de la table de droite qui ne trouve pas de correspondance dans la table de gauche, alors les colonnes de la table de gauche auront NULL comme valeurs.

### Syntaxe:

```
SELECT col1, col2, ..., coln
```

```
FROM table1 RIGHT [OUTER] JOIN table2 ON table1.id = table2.fk_id
```

### Schéma:



# I 0. Jointure

## Types de Jointure : RIGHT JOIN

### Exemple :

Compagnie		
ID_COMP	NOM_COMP	ADRESSE
AF	AIR FRANCE	Paris
SING	SINGAPORE AL	Singapour
TA	TUNISAIR	Tunis

Pilote			
NUM_PIL	NOM_PIL	NBHVOL	ID_COMPA
104	Alain Lebeau	-	-
100	Pierre Lamothe	450	AF
101	Didier Linxe	800	AF
102	Ali Salah	1100	TA
103	Henri Alquié	930	AF

➔ Afficher la liste des compagnies et leurs pilotes incluant les pilotes qui ne sont rattachés à aucune compagnie

```
SELECT NOM_COMP, NOM_PIL  
FROM COMPAGNIE RIGHT JOIN PILOTE  
ON ID_COMP = ID_COMPA ;
```

NOM_COMP	NOM_PIL
AIR FRANCE	Pierre Lamothe
AIR FRANCE	Didier Linxe
AIR FRANCE	Henri Alquié
TUNISAIR	Ali Salah
-	Alain Lebeau

# I 0. Jointure

## Types de Jointure : FULL JOIN

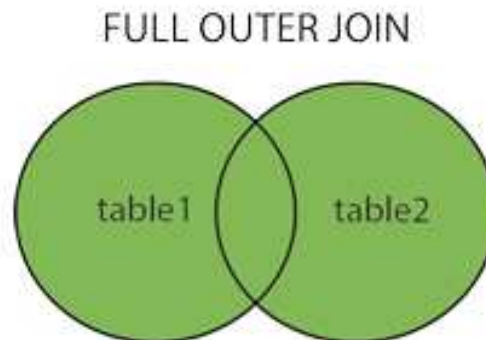
- ▶ La commande **FULL JOIN** (ou **FULL OUTER JOIN**) permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

### Syntaxe:

```
SELECT col1, col2, ..., coln
```

```
FROM table1 FULL [OUTER] JOIN table2 ON table1.id = table2.fk_id ;
```

### Schéma:



# I 0. Jointure

## Types de Jointure : FULL JOIN

### Exemple :

Compagnie		
ID_COMP	NOM_COMP	ADRESSE
AF	AIR FRANCE	Paris
SING	SINGAPORE AL	Singapour
TA	TUNISAIR	Tunis

Pilote			
NUM_PIL	NOM_PIL	NBHVOL	ID_COMPA
104	Alain Lebeau	-	-
100	Pierre Lamothe	450	AF
101	Didier Linxe	800	AF
102	Ali Salah	1100	TA
103	Henri Alquié	930	AF

→ Afficher la liste des compagnies et leurs pilotes incluant les compagnies n'ayant pas de pilotes et les pilotes qui ne sont rattachés à aucune compagnie

```
SELECT NOM_COMP, NOM_PIL
FROM COMPAGNIE FULL JOIN PILOTE
ON ID_COMP = ID_COMPA ;
```



NOM_COMP	NOM_PIL
-	Alain Lebeau
AIR FRANCE	Pierre Lamothe
AIR FRANCE	Didier Linxe
TUNISAIR	Ali Salah
AIR FRANCE	Henri Alquié
SINGAPORE AL	-

# I 0. Jointure

## Types de Jointure : CROSS JOIN

---

- ▶ CROSS JOIN retourne le produit cartésien.
- ▶ Elle permet de retourner chaque ligne d'une table avec toutes les lignes d'une autre table.
- ▶ Ainsi effectuer le produit cartésien d'une table A qui contient 30 lignes avec une table B de 40 lignes va produire 1200 lignes ( $30 \times 40 = 1200$ )
- ▶ **Attention** : Le nombre de lignes peut facilement être très élevé si l'opération est effectuée sur des tables avec un grand nombre d'enregistrements, cela peut ralentir sensiblement le serveur

### Syntaxe 1:

```
SELECT *  
FROM table1 CROSS JOIN table2 ;
```

### Syntaxe 2:

```
SELECT *  
FROM table1, table2;
```

### Exemple :

(voir slide 28)



# I 0. Jointure

## Types de Jointure : SELF JOIN (auto-jointure)

- ▶ Elle consiste à faire un rapprochement d'une table avec elle même ; c'est à dire ramener sur une même ligne des informations qui proviennent de plusieurs lignes de la même table.

**Exemple : PRODUIT ( CODE\_PDT, DES\_PDT, PU\_PDT, QTE\_STK )**

Donner la désignation des produits dont la quantité en stock est supérieure ou égale à celle du produit 'ordinateur'.

```
SELECT PI.DES_PDT
FROM PRODUIT PI JOIN PRODUIT P2
ON PI.QTE_STK >= P2.QTE_STK
WHERE P2.DES_PDT = 'ordinateur' ;
```

ou

```
SELECT PI.DES_PDT
FROM PRODUIT PI, PRODUIT P2
WHERE P2.DES_PDT = 'ordinateur'
AND PI.QTE_STK >= P2.QTE_STK;
```

Autre méthode

```
SELECT DES_PDT
FROM PRODUIT
WHERE QTE_STK >= ( SELECT QTE_STK
                   FROM PRODUIT
                   WHERE DES_PDT = 'ordinateur' ) ;
```

## II. Opérateurs ensemblistes : Union, Intersect et Minus

- ▶ La forme générale de ces opérateurs est :

```
SELECT liste_colonnes FROM table1
```

**Opérateur**

```
SELECT liste_colonnes FROM table2;
```

- ▶ Pour tous les opérateurs : **Union, Intersect et Minus**, il est à noter que :
  - ▶ Dans les deux requêtes SELECT, Il faut utiliser le même nombre de colonnes avec les mêmes types et dans le même ordre
  - ▶ Les doublons sont automatiquement éliminés
  - ▶ Les titres des colonnes résultats sont ceux de la 1<sup>ère</sup> requête
  - ▶ La clause **ORDER BY** fait référence aux noms des colonnes de la 1<sup>ère</sup> requête ou bien à leurs numéros (1,2, ...)



# 11. Opérateurs ensemblistes : Union

---

```
SELECT liste_colonnes FROM table1  
UNION  
SELECT liste_colonnes FROM table2 ;
```

- ▶ L'opérateur UNION est utilisé pour combiner le jeu de résultats de deux requêtes SELECT
- ▶ **Remarque :** UNION élimine les doublons. Pour autoriser les valeurs en double, on utilise UNION ALL

# II. Opérateurs ensemblistes : Union

---

## Exemple :

Soit la table suivante:

**ETUDIANT** (MAT\_ET, NOM\_ET, PRE\_ET, VILLE, CLASSE, GROUPE)

- ▶ Donner les noms des étudiants du groupe G1 et ceux du groupe G2 ?

```
SELECT NOM_ET  
FROM ETUDIANT  
WHERE GROUPE = 'G1'
```

**UNION**

```
SELECT NOM_ET  
FROM ETUDIANT  
WHERE GROUPE = 'G2';
```

**Select nom\_et  
from etudiant Where groupe in ('G1', 'G2');**



## II. Opérateurs ensemblistes : Intersect

---

```
SELECT liste_colonnes FROM table1  
INTERSECT  
SELECT liste_colonnes FROM table2 ;
```

- ▶ L'opérateur **INTERSECT** permet d'obtenir l'**intersection** des résultats de deux requêtes SELECT ➔ récupérer les enregistrements communs à 2 requêtes

# 11. Opérateurs ensemblistes : Intersect

---

## Exemple :

Prenons l'exemple de 2 magasins appartenant au même groupe. Chaque magasin possède sa table de clients.

**Client\_magasin1** (id, prenom, nom, ville, date\_naissance, total\_achat)

**Client\_magasin2** (id\_cli, prenom\_cli, nom\_cli, ville\_cli, date\_nais\_cli, tot\_achat\_cli)

- ▶ Donner les noms et les prénoms des clients communs aux deux magasins.

```
SELECT NOM, PRENOM
FROM CLIENT_MAGASIN1
INTERSECT
SELECT NOM_CLI, PRENOM_CLI
FROM CLIENT_MAGASIN2
ORDER BY NOM ;
```



## 11. Opérateurs ensemblistes : Minus

---

```
SELECT liste_colonnes FROM table1  
MINUS  
SELECT liste_colonnes FROM table2 ;
```

- ▶ L'opérateur **MINUS** permet d'avoir les lignes qui apparaissent dans la première requête SELECT et qui n'apparaissent pas dans la seconde.

## II. Opérateurs ensemblistes : Minus

---

### Exemple :

Soit les tables suivantes :

**Client\_inscrit** (id\_cli, nom, prenom, date\_inscription)

**Client\_refus\_email** (id\_cli, nom, prenom, date\_refus)

- ▶ Donner les noms et les prénoms des clients qui accepte de recevoir des emails informatifs

```
SELECT NOM, PRENOM  
FROM CLIENT_INSCRIT  
MINUS  
SELECT NOM, PRENOM  
FROM CLIENT_REFUS_EMAIL  
ORDER BY I ;
```