# FORMATION
# SQL et Bases de données relationnelles

Clément Hamon - 2025

# Bienvenue sur cette formation

- Clément Hamon
- Développeur depuis 5 ans
- 5 ans de SQL en entreprise
- 2 ans de formation
- clement.hamon35@gmail.com

# ib Cegos en synthèse

- 11 sites en France : La Défense, Lyon, Sophia-Antipolis, Aix-en-Provence, Toulouse, Bordeaux, Nantes, Rennes, Rouen, Lille et Strasbourg
- Une certification qualité ISO 9001, la certification Qualiopi et l'agrément Data Dock
- Offre de formations informatiques à forte valeur ajoutée depuis 35 ans
- Assure chaque année la montée en compétences de plus de 25 000 spécialistes des technologies de l'information
- Le catalogue ib Cegos : https://www.ib-formation.fr/formations

# Déroulé et structure de la formation et formalités

- Emargement le matin et l’après-midi
- Google Forms de demi-journée pour la validation des acquis et l'adaptabilité
- Évaluation du formateur le dernier jour au retour de la pause midi
- Notions théoriques suivis de mise en pratique


- Durée de la formation : 3 jours
- Objectifs :
  - Comprendre les principaux concepts des SGDBR (Système de Gestion des Bases de Données Relationnelles)
  - Appréhender l'écriture des requêtes SQL pour extraire et mettre à jour des données
  - Savoir extraire les informations de plusieurs tables
  - Assimiler les fonctions standards du langage SQL
- Organisation
  - Horaires :
    9h à 17h
  - Pauses : 15 min en matinée et après midi (vers 10h30 et 15h30)
  - Déjeuner : 1h15  (créneau 12h – 14h)


- Tour de table :
  - Prénom, nom, Société, Titre/fonction
    - Votre expérience sur le sujet de la formation ?
    - Vos attentes de la formation
    - Droits admin

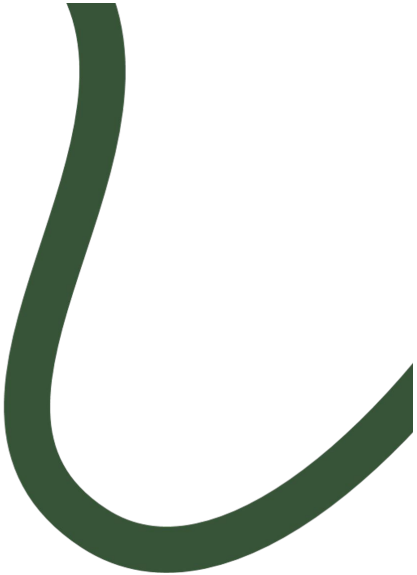
# Structure et versionning Github

- Pour chaque notion montrée, un commit sera disponible sur ce git
- https://github.com/ClementHamonDev/Formation-SQL

- Support de cours sur Teams et Github

# Introduction

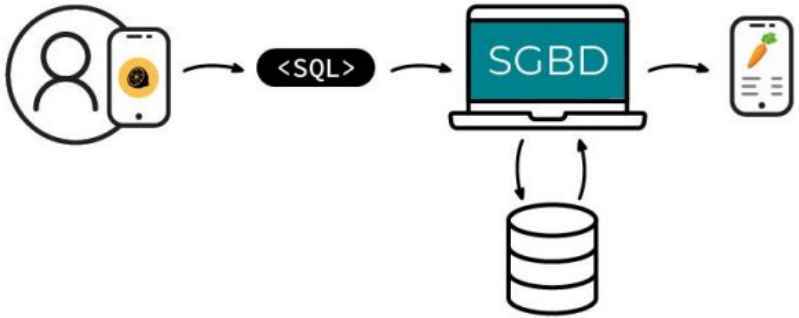
– Rappel sur le modèle relationnel



# C’est quoi une base de données

- Un ensemble organisé de données qui permet le stockage et la gestion des données, dans tous les domaines de l’informatique
- On utilisera l’acronyme BDD pour Base de Données

- Exemple : Liste de produits d’un supermarché
- Exemple : Liste de contacts d’une entreprise



# Les SGBD

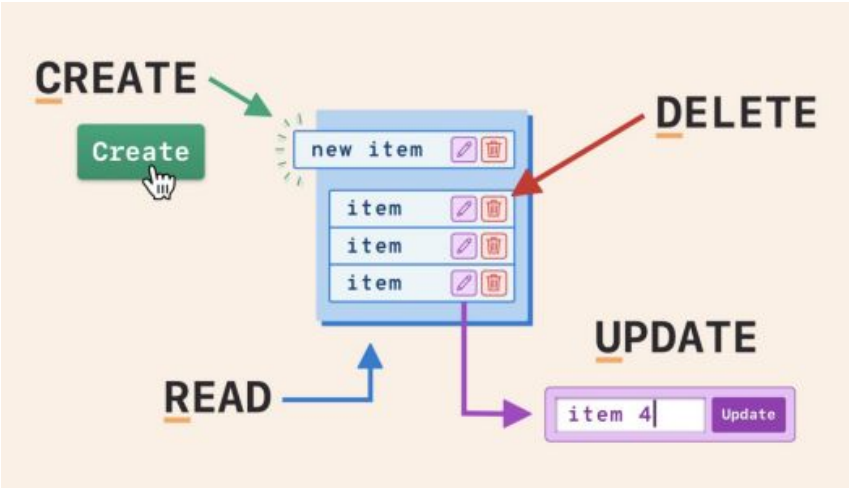
- Un SGBD (Système de Gestion de Base de Données) est un logiciel qui permet de stocker, d'organiser, de gérer et de manipuler des données dans une base de données
- Nous nous intéresserons dans ce cours aux BDD Relationnelles et donc au SGBD-R





# Persistance des bases de données

Crud pour la persistance des données



# Rappel sur le modèle relationnel

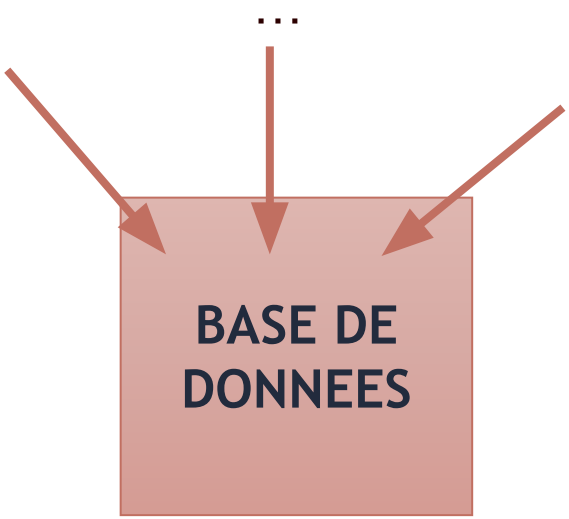
Le modèle relationnel est constitué d'un ensemble d'opérations formelles sur les relations.

Les données sont stockées dans des tables qu'on peut mettre en relation.

# Les tables

Table client
Le nom du client
L'adresse du client

Table produit
Le nom du produit
Le prix du produit




Une entité = une table

# Rappel sur le modèle relationnel

La modélisation relationnelle permet de représenter les relations à l'aide de tables

Une table représente donc une entité.

Un attribut est le nom des colonnes qui constitue la définition d'une table. Il comporte un typage de données.

On appelle tuple (ou n-uplet) une ligne de la table.

Une relation se fait entre des colonnes ayant des données qui correspondent.

# Rappel sur le modèle relationnel

La cardinalité d'une relation est le nombre de tuples qui la composent.

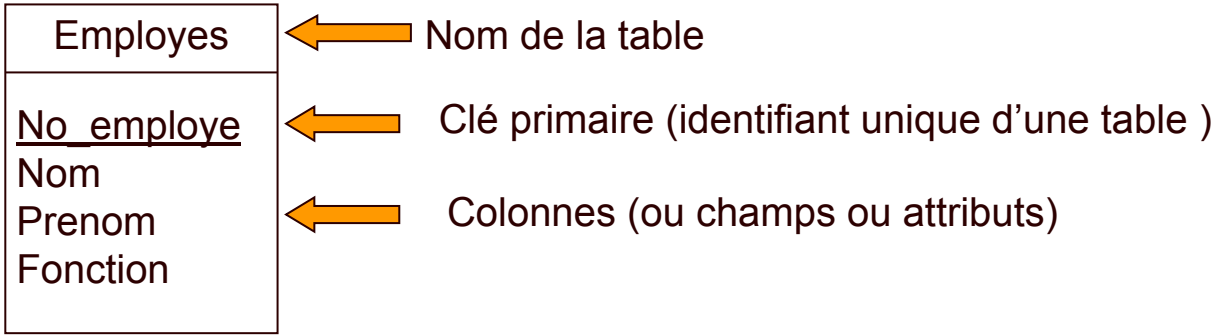
La clé primaire d'une relation est l'attribut, ou l'ensemble d'attributs, permettant de désigner de façon unique un tuple.

Une clé étrangère, par contre, est une clé faisant référence à une clé appartenant à une autre table. Cela permet de relier deux tables.

Le nom des colonnes à relier peut être différent, mais le type de données doit être le même.

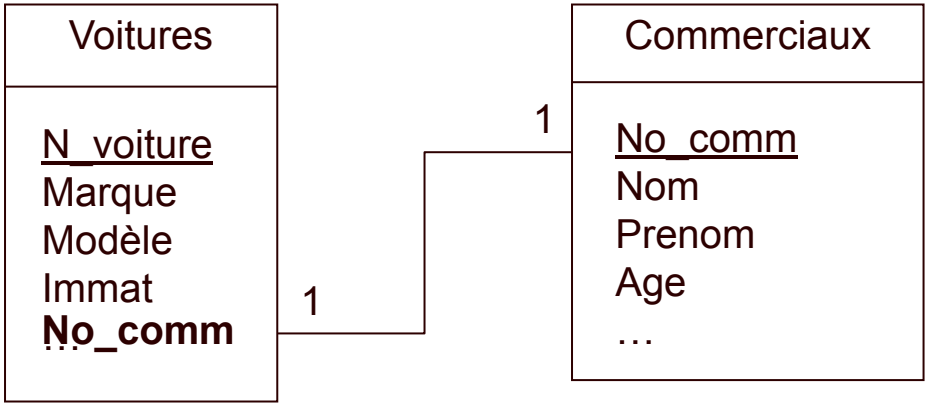
# Rappel sur le modèle relationnel

Les données sont stockées dans des tables



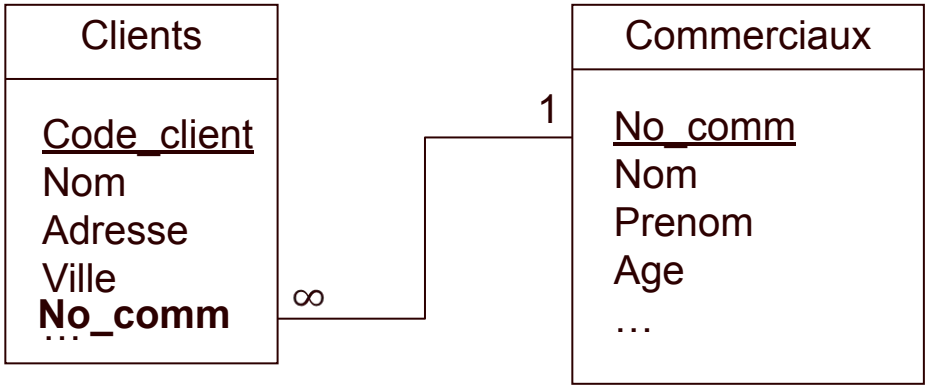
# Rappel sur le modèle relationnel

Les relations entre les tables : de un à un



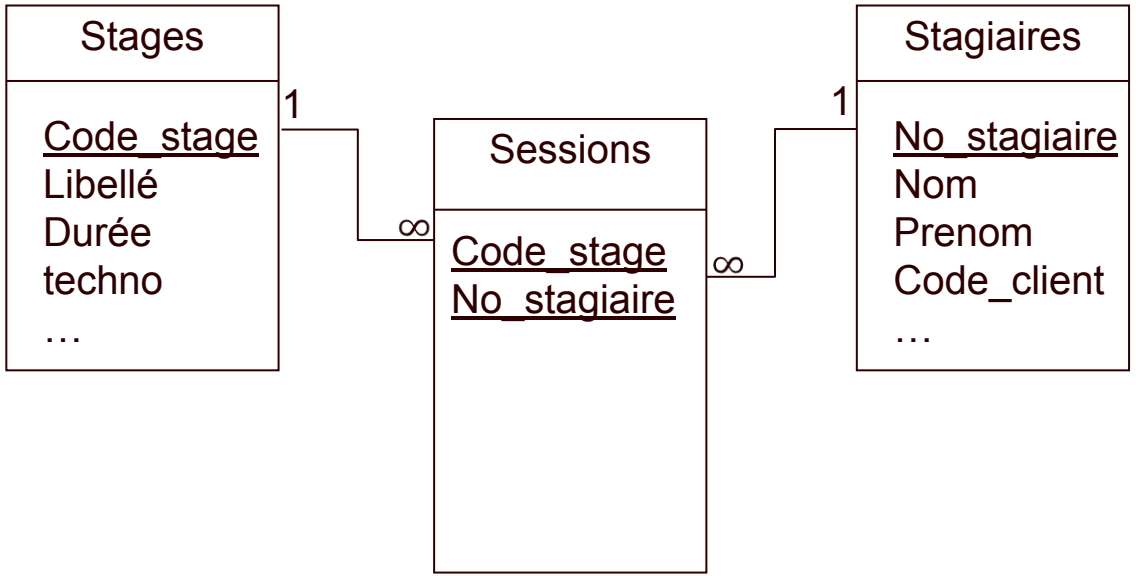
# Rappel sur le modèle relationnel

Les relations entre les tables : de un à plusieurs



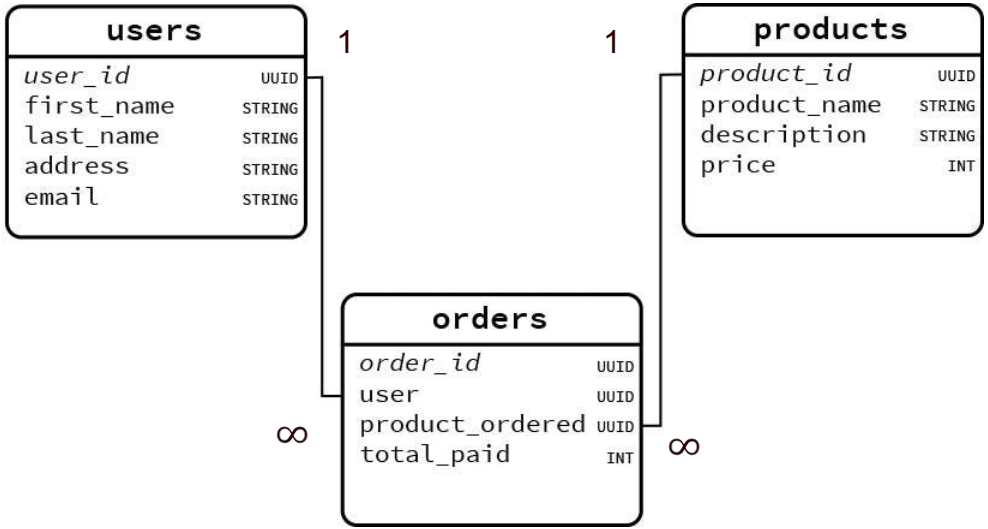
# Rappel sur le modèle relationnel

Les relations entre les tables : relation de plusieurs à plusieurs



# Atelier 1

- Interprétez de manière textuelle les cardinalités.
- Proposez un exemple de modélisation avec des données et interprétez ces données



# Atelier 1

Représentez le modèle suivant :

- Un magasin est représenté par un numéro unique, et une adresse.
- Un produit est désigné par une référence unique, un nom et un prix.
- Un magasin peut stocker aucun comme plusieurs produits, tandis qu'un produit est lui stocké dans un ou plusieurs magasins.
- Nous souhaitons également sauvegarder la date de stockage d'un produit dans notre système.

# Rappel sur le modèle relationnel

Caractéristiques :

- Indépendance physique : le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations.
- Indépendance logique : le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs .

# Rappel sur le modèle relationnel

Caractéristiques :

- Manipulabilité : des personnes ne connaissant pas la base de données doivent être capables de décrire leur requête sans faire référence à des éléments techniques de la base de données.
- Rapidité des accès : le système doit pouvoir fournir les réponses aux requêtes le plus rapidement possible, cela implique des algorithmes de recherche rapides.
- Administration centralisée : le SGBD doit permettre à l'administrateur de pouvoir manipuler les données, insérer des éléments, vérifier son intégrité de façon centralisée.

# Rappel sur le modèle relationnel

Caractéristiques :

- Limitation de la redondance : le SGBD doit pouvoir éviter dans la mesure du possible des informations redondantes, afin d'éviter d'une part un gaspillage d'espace mémoire mais aussi des erreurs .
- Vérification de l'intégrité : les données doivent être cohérentes entre elles, de plus lorsque des éléments font référence à d'autres, ces derniers doivent être présents.

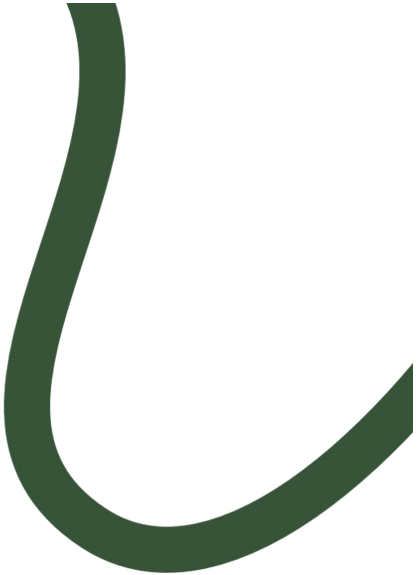
# Rappel sur le modèle relationnel

Caractéristiques :

- Partageabilité des données : le SGBD doit permettre l'accès simultané à la base de données par plusieurs utilisateurs.
- Sécurité des données : le SGBD doit présenter des mécanismes permettant de gérer les droits d'accès aux données selon les utilisateurs.

# Introduction

– Les caractéristiques du langage SQL



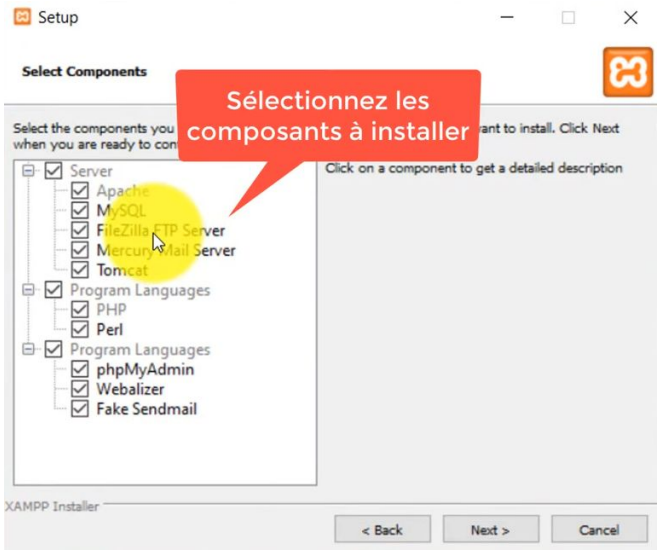
# Langage SQL : les caractéristiques

Le langage Sql est devenu le standard en matière d'interface relationnelle, ceci probablement parce que ce langage est :

- issu de SEQUEL (interface de System-R), SQL a été développé chez IBM à San José
- basé sur des mots clefs anglais explicites, il est relativement simple et facile à apprendre.
- normalisé

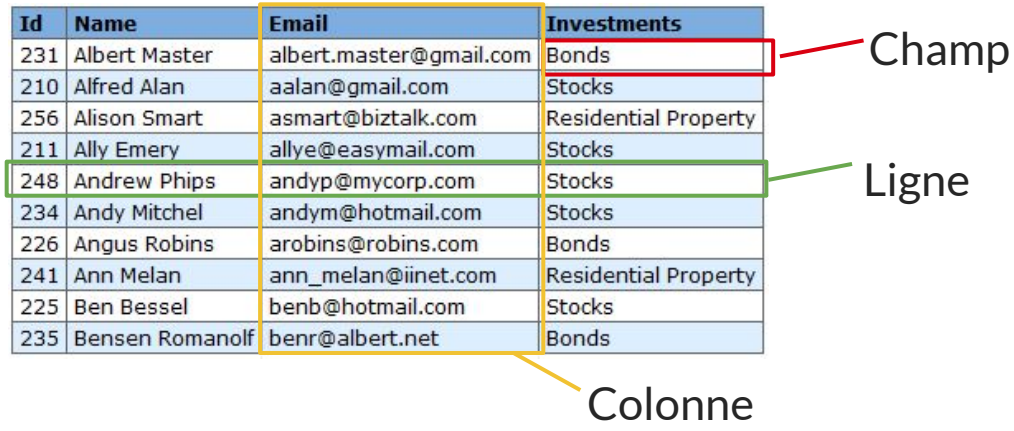
# Installation de MYSQL, PHPMYADMIN

1. https://www.apachefriends.org/fr/index.html
2. Durant l'installation, laisser cocher phpmyadmin et mysql
3. http://127.0.0.1/dashboard/



# SQL

- Un entité sera représentée par une table
- Un table contient des colonnes qui sont nommées par des noms. Ces noms sont l'équivalent des attributs.
- Une ligne, ou un "tuple", représente une entrée dans la table.
- Un champ correspond à une valeur précise de ma table




- Clé primaire et clé secondaire
- Relations entre les tables
- Intégrité référentielle

Echanges

# Module 2
# Le Langage d'Interrogation de Données (LID)

# Le Langage d’Interrogation des Données (LID) : les bases

- La Sélection de données

# La commande SELECT

Le SELECT est la commande de base du SQL .

Elle permet d’extraire des données d’une base ou  d’en calculer de nouvelles à partir de données  existantes .

# La commande SELECT

*SELECT* * ou liste de colonnes

*FROM* nom de table

# La commande SELECT

Requête:

```
SELECT CLI_NOM, CLI_PRENOM FROM T_CLIENT;
```

Résultat :

```
CLI_NOM
CLI_PRENOM
-------   ----------
DUPONT Alain
MARTIN Marc
 ...
```

# La commande SELECT

Il existe plusieurs opérateurs de sélection :

- Le caractère * récupère toutes les colonnes

  select * from nom_table;

- DISTINCT permet d'éliminer les doublons dans la réponse

  select distinct nom_colonne from nom_table;

- AS sert à donner un nom aux colonnes renvoyées par la requête

  select nom_colonne as nom_alias from nom_table;

# La commande SELECT

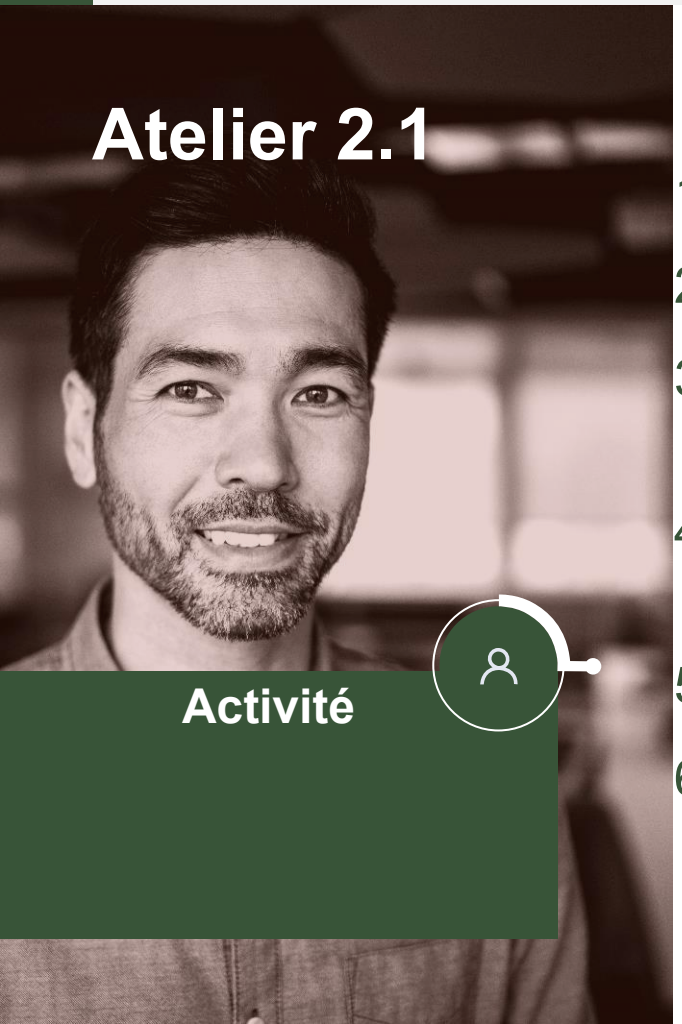
Il existe plusieurs opérateurs de sélection :

- On peut utiliser les opérateurs mathématiques de base pour combiner différentes colonnes (+,-, *, /)

  select nom_colonne1*nom_colonne2 from nom_table;

- L'opérateur || (double barre verticale) permet de concaténer des champs de type caractères (fonction concat ou le signe + dans certaines bases de données)

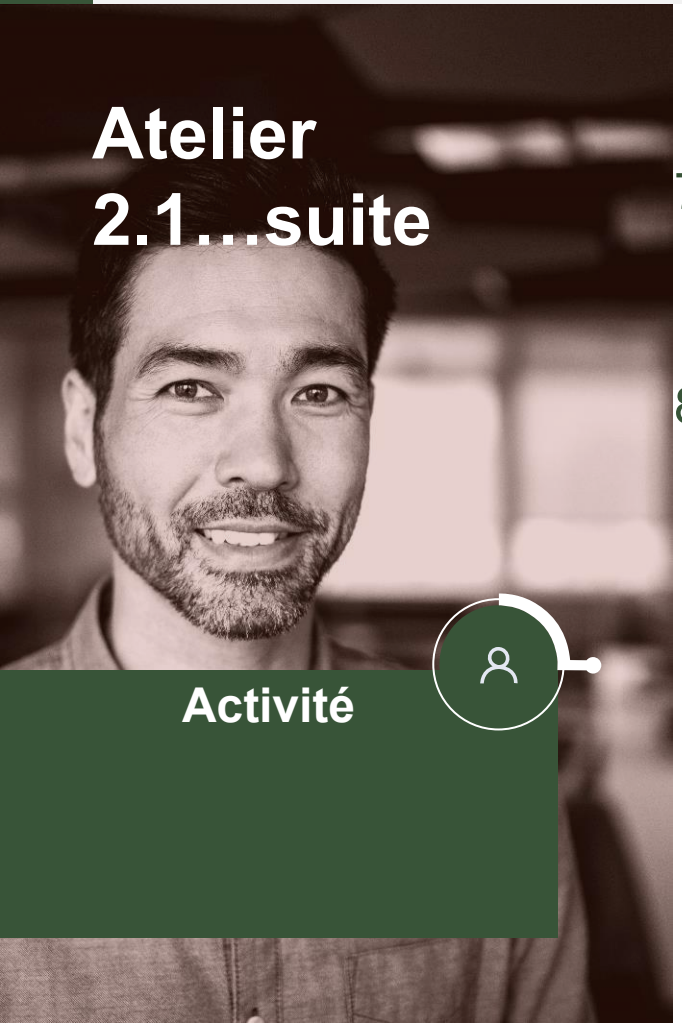
# Atelier 2.1



**Activité**

Je reste disponible pour toute question

1. Affichez tous les employés de la société.
2. Affichez le nom des catégories de produits.
3. Affichez le nom, le prénom, la date de naissance et la commission
4. Affichez la liste des fonctions des employés de la société.
5. Affichez la liste des pays de nos clients.
6. Affichez la liste des localités dans lesquelles il existe au moins un client.



# Atelier 2.1…suite

Activité

Je reste disponible pour toute question

7. Affichez les produits commercialisés et la valeur de stock par produit ( prix unitaire * unités en stock).
8. Affichez le nom, le prénom, l’âge et l’ancienneté des employés.

# Ce qu’il faut retenir

- Respecter l’ordre d’introduction des clauses : select…from
- Le select peut contenir trois types d’expression : une colonne, une constante, un calcul
- Penser à la problématique des valeurs null

# Le Langage d’Interrogation des Données (LID) : les bases

- Les restrictions ou conditions

# La clause WHERE

Cette clause permet d’afficher des lignes vérifiant une condition.

Syntaxe :

SELECT ....

FROM ....

WHERE *condition*

# La clause WHERE

Dans la clause Where (clause conditionnelle), on utilise des opérateurs logiques.

Where Expression opérateur Expression

L'opérateur peut être:

- =
- <, <=
- >, >=
- !=, <>,^=

Ex: Select * from client where numclient=123;

# La clause WHERE

Il existe d'autres opérateurs :

- Opérateur IN / NOT IN

  WHERE VILLE IN ('Bordeaux','Dax','Biarritz')

- Opérateur BETWEEN / NOT BETWEEN

  WHERE PRIX BETWEEN 30 AND 50

- Opérateur LIKE / NOT LIKE

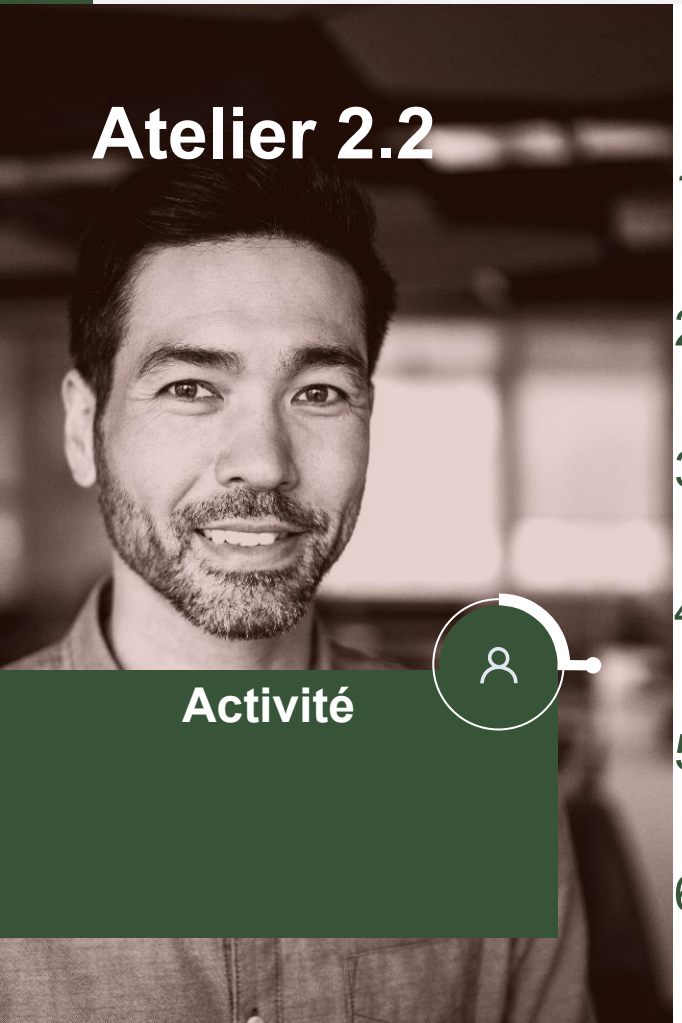
  WHERE CLI_NOM LIKE 'B%'

- Comparaison logique

  IS [NOT] {TRUE | FALSE} / IS [NOT] NULL

- Connecteurs logiques : permettent de poser plusieurs conditions dans la clause where

  OR | AND



# Atelier 2.2

**Activité**

Je reste disponible pour toute question

1. Affichez le nom de la société et le pays des clients qui habitent à Toulouse.
2. Affichez le nom, le prénom et la fonction des employés dirigés par l'employé numéro 2.
3. Affichez le nom, le prénom et la fonction des employés qui ne sont pas des représentants.
4. Affichez le nom, le prénom et la fonction des employés qui ont un salaire inférieur à 3500.
5. Affichez le nom, la ville et le pays des clients qui n'ont pas de fax.
6. Affichez le nom, le prénom et la fonction des employés qui n'ont pas de supérieur.

# Ce qu’il faut retenir

## Echanges

- Mettre les valeurs de type texte et date entre simples quotes (apostrophes)
- Chez certains éditeurs de bdd, le where est sensible à la casse des valeurs
- Si on utilise, dans le même where, à la fois plusieurs or et and  and est prioritaire. Utiliser des parenthèses pour forcer la priorité.

# Le Langage d’Interrogation des Données (LID) : les bases

- Les tris

# La clause ORDER BY

Cette clause permet de définir le tri des colonnes.

ASC spécifie l’ordre ascendant et DESC l’ordre descendant du tri. ASC ou DESC peut être omis. Dans ce cas c'est l'ordre ascendant qui est utilisé par défaut.

```
SELECT ...
FROM ...
WHERE ...
ORDER BY nom_colonne;
```

```
SELECT ...
FROM ...
WHERE ...
ORDER BY nom_colonne desc;
```

# La clause ORDER BY

Il est possible de trier, successivement, sur plusieurs colonnes.

```
SELECT …
FROM ...
WHERE ...
ORDER BY colonneA, colonneB;
```

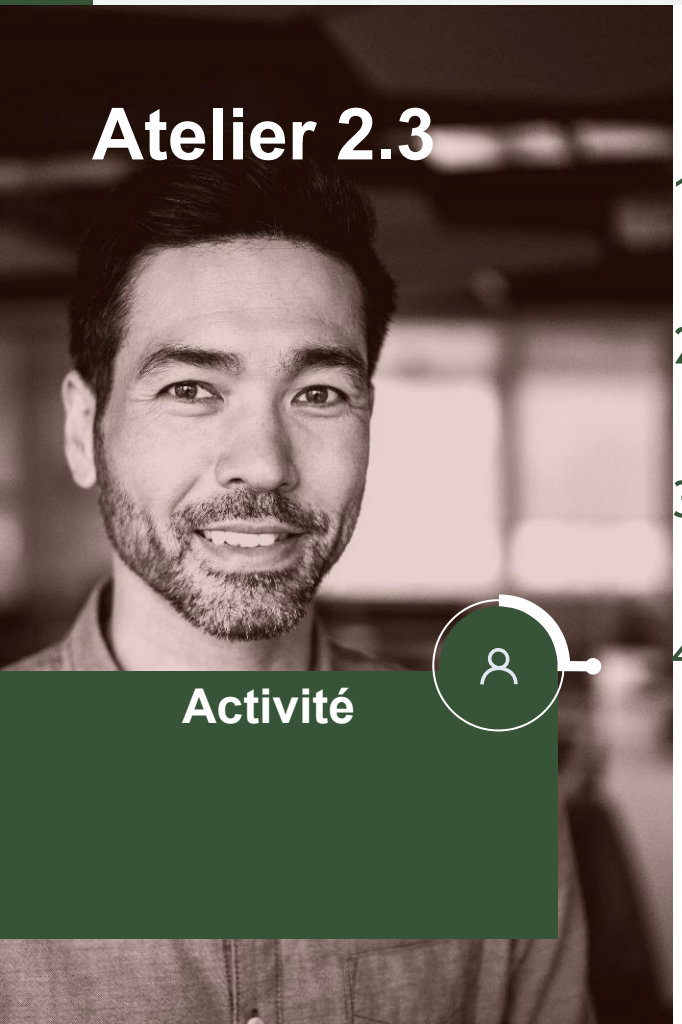
On peut dans cette clause faire référence à la position de cette colonne dans le select.
Ici on trie d'abord sur la colonne B en décroissant, puis sur la colonne A de façon croissante

```
SELECT colonneA, colonneB
FROM ....
WHERE ....
ORDER BY 2 desc, 1;
```

# La clause ORDER BY

On peut utiliser dans le order by un alias défini dans le select.

```
SELECT colonneA,
       colonneB as alias
FROM ...
WHERE ...
ORDER BY alias;
```



# Atelier 2.3

Activité

Je reste disponible pour toute question

1. Afficher le nom et prénom des employés, triés par nom en ordre décroissant.
2. Afficher le nom et la ville des clients, triés par pays.
3. Afficher le nom, le pays et la ville des clients, triés par pays et par ville.
4. Afficher le nom et la commission des employés, triés par commission.

# Ce qu’il faut retenir

- La clause order by se saisit toujours à la fin de la requête
- Clause qui impacte de façon conséquente le temps de réponse des requêtes

# Le Langage d’Interrogation des Données (LID) : les bases

- Les jointures

# Jointures dans le where

## Jointures entre deux tables.

Soit deux tables : *table1* et *table2*. *table1* a les colonnes *col1* et *col2* et *table2* les colonnes *cola*, *colb*.

Supposons que le contenu des tables soit le suivant :

| table1 | col1 | col2 |
|---|---|---|
| | x | 3 |
| | y | 4 |

| table2 | cola | colb |
|---|---|---|
| | a | 7 |
| | b | 4 |

Soit la commande :

SELECT *col1, cola* FROM *table1, table2*

WHERE *table1.col2=table2.colb;*

Cette requête extrait des données de deux tables : *table1* et *table2* avec une condition entre deux colonnes de tables différentes.

# Jointures dans le where

## Comment fonctionne-t-elle ?

Une nouvelle table est construite avec pour colonnes l'ensemble des colonnes des deux tables et pour lignes le produit cartésien des deux tables :

| col1 | col2 | cola | colb |
|---|---|---|---|
| x | 3 | a | 7 |
| x | 3 | b | 4 |
| y | 4 | a | 7 |
| y | 4 | b | 4 |

La condition *WHERE col2=colb* est appıiquée à cette nouvelle table. On obtient donc la nouvelle table suivante :

| col1 | col2 | cola | colb |
|---|---|---|---|
| y | 4 | b | 4 |

# Jointures

Il y a ensuite affichage des colonnes demandées (select) :

| col1 | cola |
|---|---|
| y | b |

# Jointures dans le where

Syntaxe d'une requête multi-tables :

SELECT *colonne1, colonne2, ...*

FROM *table1, table2, ..., tablep*

WHERE *jointure1*

AND *jointure2*

AND *…*

ORDER BY ...;

# Jointures dans le where

- Sans condition ( produit cartésien) :

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE;
```

- Avec condition :

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE
WHERE T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID;
```

En renommant les tables et avec une condition supplémentaire dans le where :

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT C, T_TELEPHONE T
WHERE C.CLI_ID = T.CLI_ID AND TYP_CODE = 'FAX‘;
```

# Jointures avec un join

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT NATURAL JOIN T_TELEPHONE;
```

La jointure se fait sur la colonne qui porte le même nom dans les deux tables. Et une seule colonne doit avoir le même nom de part et d'autre des tables

| | |
|---|---|
| **Jointure naturelle** | **SELECT ...**<br>**FROM <table gauche>**<br>**NATURAL JOIN <table droite>**<br>; |

# Jointures avec un join

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT JOIN T_TELEPHONE USING (CLI_ID);
```

La jointure se fait sur la colonne qui porte le même nom dans les deux tables.

| | |
|---|---|
| **Jointure join…using** | **SELECT ...**<br>**FROM <table gauche>**<br>**[INNER] JOIN <table droite>**<br>**USING (nom de la colonne)**<br>**;** |

# Jointures avec un join

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT INNER JOIN T_TELEPHONE
ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID;
```

Ici on spécifie le nom des colonnes sur lesquelles se fait la jointure.

| | |
|---|---|
| **Jointure interne** | **SELECT ...**<br>**FROM <table gauche>**<br>**[INNER] JOIN <table droite>**<br>**ON <condition de jointure>**<br>**;** |

# Jointures avec un join

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT C LEFT OUTER JOIN T_TELEPHONE T
ON C.CLI_ID = T.CLI_ID;
```

Les jointures externes rapatrient les informations disponibles, même si des lignes de la table ne sont pas renseignées entre les différentes tables jointes

| | |
|---|---|
| **Jointure externe** | **SELECT ...**<br>**FROM <table gauche>**<br>**LEFT \| RIGHT \| FULL [OUTER] JOIN <table droite>**<br>**ON condition de jointure** |

# Jointures externe gauche

SELECT colonnes

FROM TGauche LEFT OUTER JOIN TDroite

ON condition de jointure;

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans le prédicat (équivalent de l’inner join), puis on rajoute toutes les lignes de la table TGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.

# Jointures externe droite

SELECT colonnes

FROM TGauche RIGHT OUTER JOIN TDroite

ON condition de jointure

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat (équivalent de l’inner join), puis on rajoute toutes les lignes de la table TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

# Jointures externe complète

SELECT colonnes

FROM TGauche FULL OUTER JOIN TDroite

ON condition de jointure

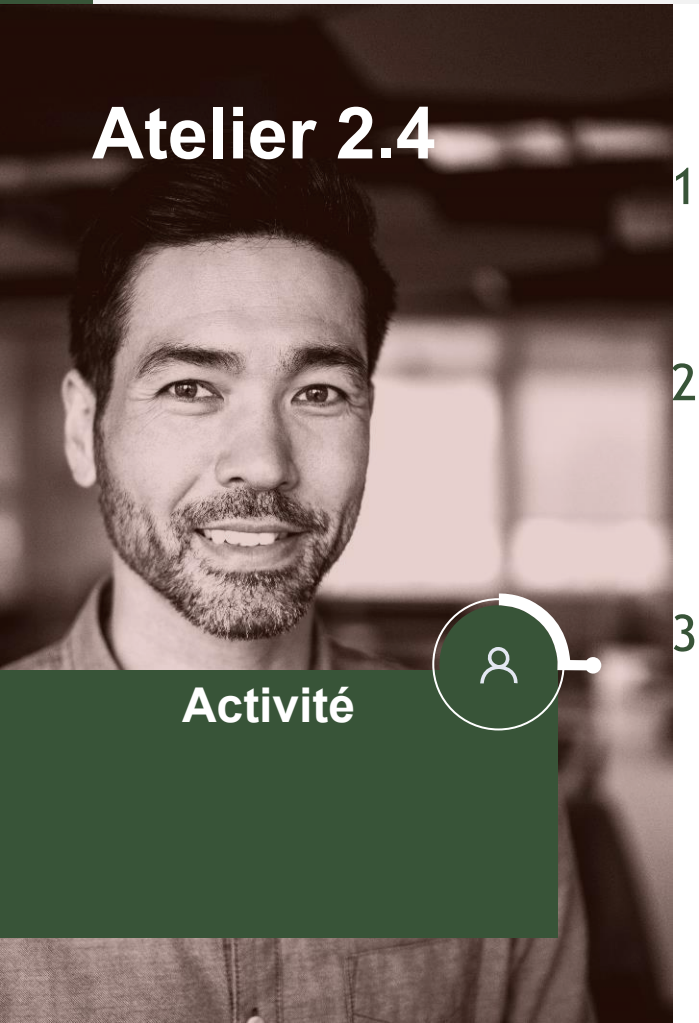
On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat (équivalent de l’inner join), puis on rajoute toutes les lignes de la table TGauche et TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

# Jointures croisée

```
SELECT CHB_ID, TRF_DATE_DEBUT, 0 AS TRF_CHB_PRIX
FROM T_TARIF CROSS JOIN T_CHAMBRE
ORDER BY CHB_ID, TRF_DATE_DEBUT;
```

On fait sciemment le produit cartésien .

| | |
|---|---|
| **Jointure croisée** | **SELECT ...**<br>**FROM <table gauche>**<br>**CROSS JOIN <table droite>** |

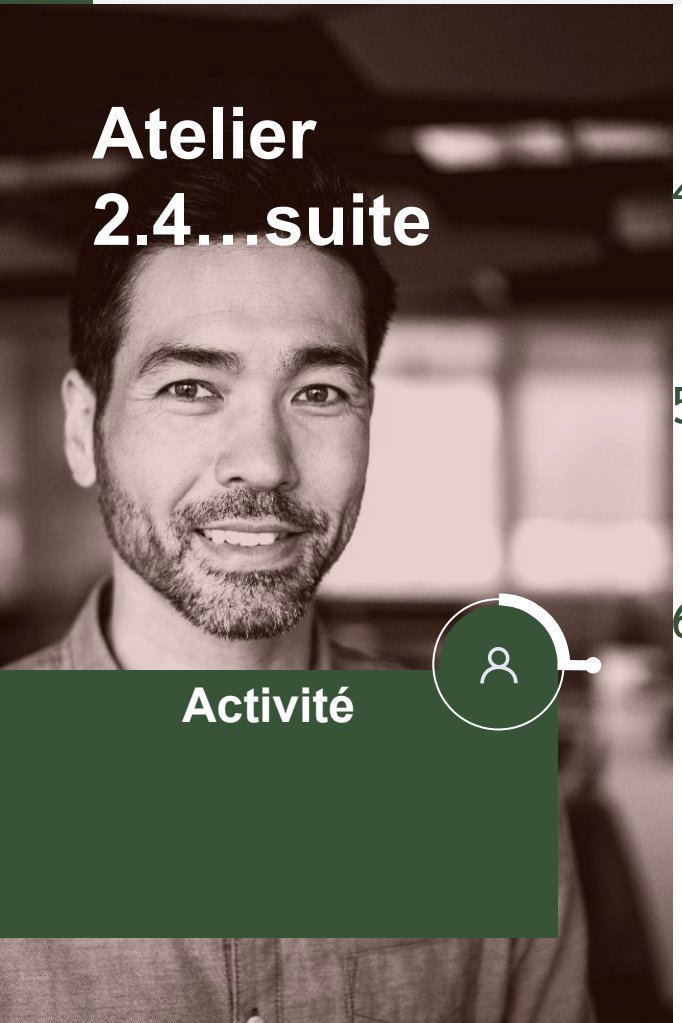


# Atelier 2.4

**Activité**

Je reste disponible pour toute question

1. Affichez le nom, le prénom, la fonction et le salaire des employés qui ont un salaire compris entre 2500 et 3500.
2. Affichez le nom du produit, le nom du fournisseur, le nom de la catégorie et les quantités de produits qui ne sont pas d'une des catégories 1,3,5 et 7.
3. Affichez le nom du produit, le nom du fournisseur, le nom de la catégorie et les quantités des produits qui ont un numéro de fournisseur entre 1 et 3 ou un code catégorie entre 1 et 3, et pour lesquelles les quantités sont données en boîte(s) ou en carton(s).



# Atelier 2.4…suite

Activité

Je reste disponible pour toute question

4. Écrivez la requête qui permet d'afficher le nom des employés qui ont effectué au moins une vente pour un client parisien.
5. Affichez le nom des produits et le nom des fournisseurs pour les produits des catégories 1,4 et 7.
6. Affichez le nom des employés ainsi que le nom de leur supérieur hiérarchique

# Ce qu’il faut retenir

- Sans jointures, on obtient le produit cartésien des tables (résultat incohérent).
- Le join est plus utilisé que le where.
- Les 2 colonnes communes effectuant la jointure entre 2 tables doivent avoir le même type de données. Pas forcément le même nom.

# Module 3
# Utilisation de fonctions

# Fonctions de chaînes de caractères

## Extraire une sous chaîne :

La fonction SUBSTRING (SUBSTR sous Oracle et Postgre) permet d'extraire une sous chaîne d'une chaîne de caractère.

Exemple SQL Server :

```
SELECT CLI_NOM, CLI_PRENOM, SUBSTRING(CLI_PRENOM,1,1) + SUBSTRING
(CLI_NOM,1,1) AS INITIALES FROM T_CLIENT;
```

| CLI_NOM | CLI_PRENOM | INITIALES |
|---|---|---|
| DUPONT | Alain | AD |
| MARTIN | Marc | MM |
| …… | | |

# Fonctions de chaînes de caractères

| | |
|---|---|
| CONCAT(expression1,expression2,…) | concaténation : utiliser de préférence \|\| chez Oracle (car deux arguments maximum avec la fonction) |
| INITCAP(expression)<br>(Oracle, Postgre) | initiales en lettres capitales |
| LPAD(expression,nb_caractères,caractère)<br>(Mysql ,Oracle, Postgre) | complément ou troncature à n position à gauche |
| RPAD(expression,nb_caractères,caractère)<br>(Mysql ,Oracle, Postgre) | complément ou troncature à n position à droite |
| LTRIM(expression)<br>RTRIM(expression)<br>TRIM(expression) | suppression des espaces en tête/queue d'une chaîne |
| REPLACE (expression, valeur à remplacer, valeur de remplacement) | remplacement |
| INSTR(expression,'texte cherché')<br>(Mysql ,Oracle) | position d'une chaîne dans une sous chaîne |
| PATINDEX('%texte cherché%', expression)<br>(Sql Server) | position d'une chaîne dans une sous chaîne |
| POSITION('texte cherché' in expression)<br>(Postgre) | position d'une chaîne dans une sous chaîne |

# Fonctions de chaînes de caractères

| | |
|---|---|
| LENGTH(expression)<br>(Mysql ,Oracle, Postgre) | longueur de la chaîne |
| LEN (expression)<br>( Sql Server) | longueur de la chaîne |
| ASCII(expression) | code ASCII d'un caractère |
| CHR(valeur)<br>(Oracle et Postgre) | caractère dont le code ASCII est donné |
| CHAR(valeur)<br>(Mysql et Sql Server) | caractère dont le code ASCII est donné |
| REVERSE(expression) | Inverse l'ordre des caractères d'une chaîne |

# Fonctions date système

Heure et date courante :

Exemple Postgre, MySql, Oracle :

```
SELECT NO_COMMANDE
FROM COMMANDES WHERE
DATE_ENVOI BETWEEN CURRENT_DATE
and CURRENT_DATE - 14;
```

| | |
|---|---|
| Oracle | SYSDATE<br>CURRENT_DATE |
| MySQL<br>Postgre | CURRENT_DATE |
| SQL Server | GETDATE() |
| Access | DATE() |

# Fonctions de dates

| | |
|---|---|
| Expression + INTERVAL ‘1’ day (Mysql ,Oracle, Postgre) | ajoute des jours, des mois, des années à une date |
| DATEADD (day,1,expression) (Sql Server) | ajoute des jours, des mois, des années à une date |
| ADD_MONTHS(expression, nb_mois) (Oracle) | ajoute des mois à une date |
| NEXT_DAY (expression,’jour’) (Oracle) | date du prochain jour d’un nom donné |
| LAST_DAY(expression) (Oracle, Mysql) | renvoie le n° du dernier jour d’un mois d’une date |
| MONTHS_BETWEEN(date1,date2) (Oracle) | nombre de mois entre deux dates |
| DATEDIFF(day \| month \| year, date_début,date_fin) (Sql Server)<br>DATEDIFF(date_début,date_fin) (Mysql) | différence entre deux dates |
| DATEPART(day \| month \| year, expression) (Sql Server) | Renvoie la valeur du jour, mois ou année |
| EXTRACT(day \| month \| year from expression) (Mysql ,Oracle, Postgre) | Renvoie la valeur du jour, mois ou année |

# Fonctions d’agrégation (ou de regroupement)

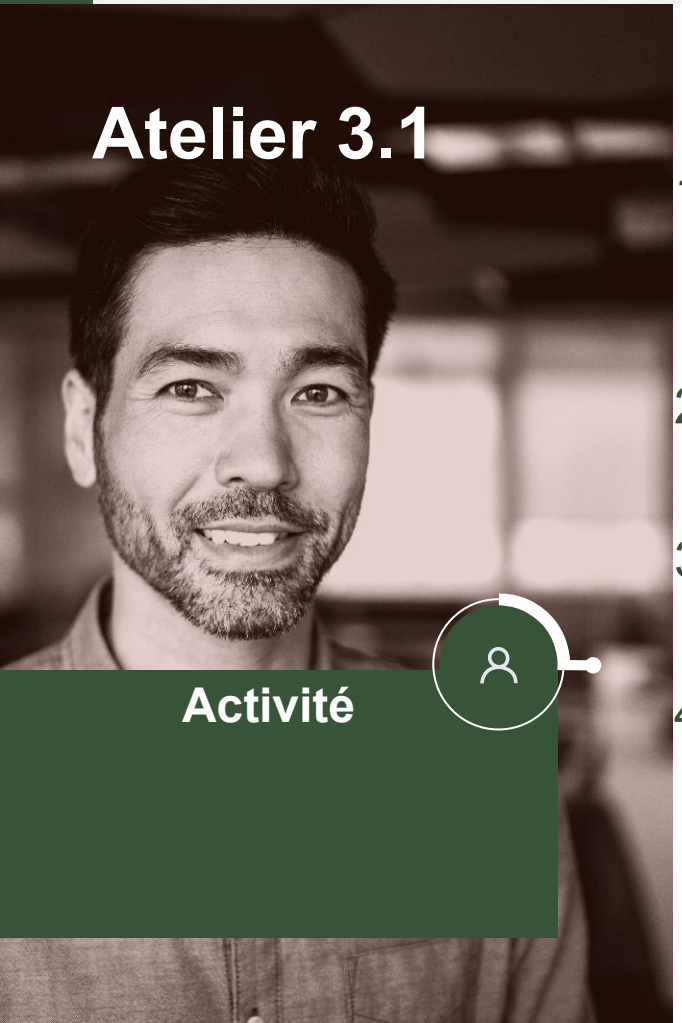
Fonctions statistiques :

```
SELECT AVG(TRF_CHB_PRIX) as MOYENNE,
MAX(TRF_CHB_PRIX) as MAXI,
MIN(TRF_CHB_PRIX) as MINI,
SUM(TRF_CHB_PRIX) as TOTAL,
COUNT(TRF_CHB_PRIX) as NOMBRE
FROM TJ_TRF_CHB ;
```

# Fonctions mathématiques

| | |
|---|---|
| ABS(expression) | valeur absolue |
| MOD(expression) | modulo |
| SIGN(expression) | signe |
| SQRT(expression) | racine carrée |
| CEIL(expression) ; Sql Server : ceiling(expression) | plus grand entier |
| FLOOR(expression) | plus petit entier |
| ROUND(expression, nb_décimales) | arrondi |
| TRUNC(expression, nb_décimales)<br>Sql Server : ROUND(expression, nb_décimales,1) | tronqué |
| EXP(expression) | exponentielle |
| LN(expression) | logarithme népérien |
| LOG(expression) | logarithme décimal |
| POWER(expression,valeur) | puissance |
| COS(expression) ; SIN((expression) | cosinus ; sinus |
| TAN(expression) | tangente |
| PI() ; Oracle : ASIN(1)*2 | constante Pi |

# Atelier 3.1



**Activité**

Je reste disponible pour toute question

1. Affichez la somme des salaires et des commissions des employés (cumuler les 2 en une seule colonne).
2. Affichez la moyenne des salaires et la moyenne des commissions des employés.
3. Affichez le salaire maximum et la plus petite commission des employés.
4. Affichez le nombre distinct de fonction.

# La clause GROUP BY

La clause GROUP BY est nécessaire dès que l'on utilise des fonctions de calculs statistiques avec des données brutes. Cette clause groupe les lignes en se basant sur la valeur de colonnes spécifiées et renvoie une seule ligne par groupe.

# La clause GROUP BY

Sans group by :

SELECT COUNT(CHB_ID) AS NOMBRE, CHB_ETAGE FROM T_CHAMBRE;

NOMBRE CHB_ETAGE
---------- ---------
1 RDC
1 RDC
1 RDC
1 RDC
1 1er
1 1er

……

# La clause GROUP BY

Avec Group By :

```
SELECT COUNT(CHB_ID) AS NOMBRE, CHB_ETAGE FROM T_CHAMBRE
GROUP BY CHB_ETAGE;
```

```
NOMBRE CHB_ETAGE
---------- ---------
8 1er
8 2e
4 RDC
```

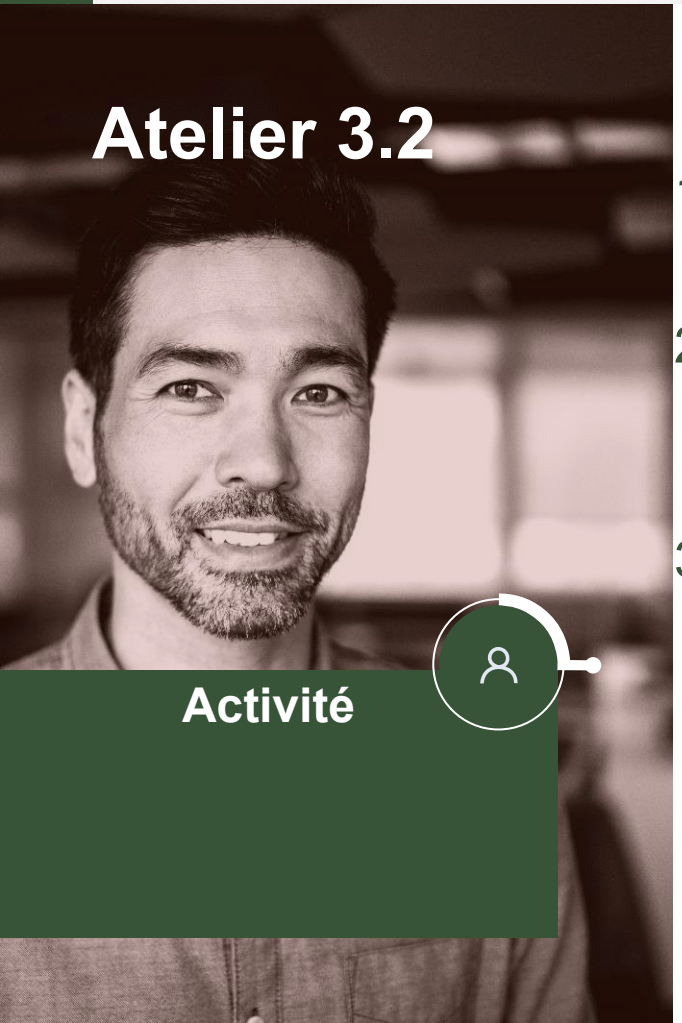
# La clause HAVING

La clause HAVING remplace le WHERE
sur les opérations résultant des regroupements

```
SELECT SUM(CHB_COUCHAGE) AS NOMBRE,
        CHB_ETAGE
FROM T_CHAMBRE
GROUP BY CHB_ETAGE
HAVING SUM(CHB_COUCHAGE) >= 20;
```

```
NOMBRE CHB_ETAGE
---------- ---------
23 1er
22 2e
```

# Atelier 3.2



Activité

Je reste disponible pour toute question

1. Écrivez la requête qui permet d’afficher la masse salariale des employés par fonction.
2. Affichez le montant de chaque commande, en ne conservant que les commandes qui comportent plus de 5 références de produit.
3. Afficher la valeur des produits en stock et la valeur des produits commandés par fournisseur, pour les fournisseurs qui ont un numéro compris entre 3 et 6.

# Ce qu’il faut retenir

## Echanges

- Lorsque dans un select on se retrouve avec au moins une fonction de regroupement et au moins une expression autre = tout ce qui n'est pas fonction de regroupement doit être dans la clause group by.
- La clause having filtre en prenant en compte la clause group by.
  Alors que la clause where intervient sur les lignes du from (et des éventuelles jointures).

# Module 4
# Opérateurs ensemblistes

# Les opérateurs ensemblistes

Les opérations ensemblistes en SQL sont celles définies dans l'algèbre relationnelle. Elles sont réalisées grâce aux opérateurs :

- UNION
- INTERSECT (n'est pas implémenté dans tous les SGBD)
- EXCEPT ou MINUS (n'est pas implémenté dans tous les SGBD)

Ces opérateurs s'utilisent entre deux clauses *SELECT*.

# Les opérateurs assemblistes

L'opérateur UNION :

Cet opérateur permet d'effectuer une UNION des lignes sélectionnées par deux clauses SELECT.

SELECT ---- FROM ---- WHERE ------ UNION

SELECT ---- FROM ---- WHERE ------ ;

Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause UNION ALL.

# Les opérateurs ensemblistes

L'opérateur INTERSECT :

Cet opérateur permet d'effectuer une INTERSECTION des enregistrements sélectionnés par deux clauses SELECT.

SELECT ---- FROM ---- WHERE ------ INTERSECT

SELECT ---- FROM ---- WHERE ------ ;

L'opérateur INTERSECT n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles (sous-interrogation):

SELECT a,b FROM table1 WHERE EXISTS ( SELECT c,d FROM table2 WHERE a=c AND b=d );

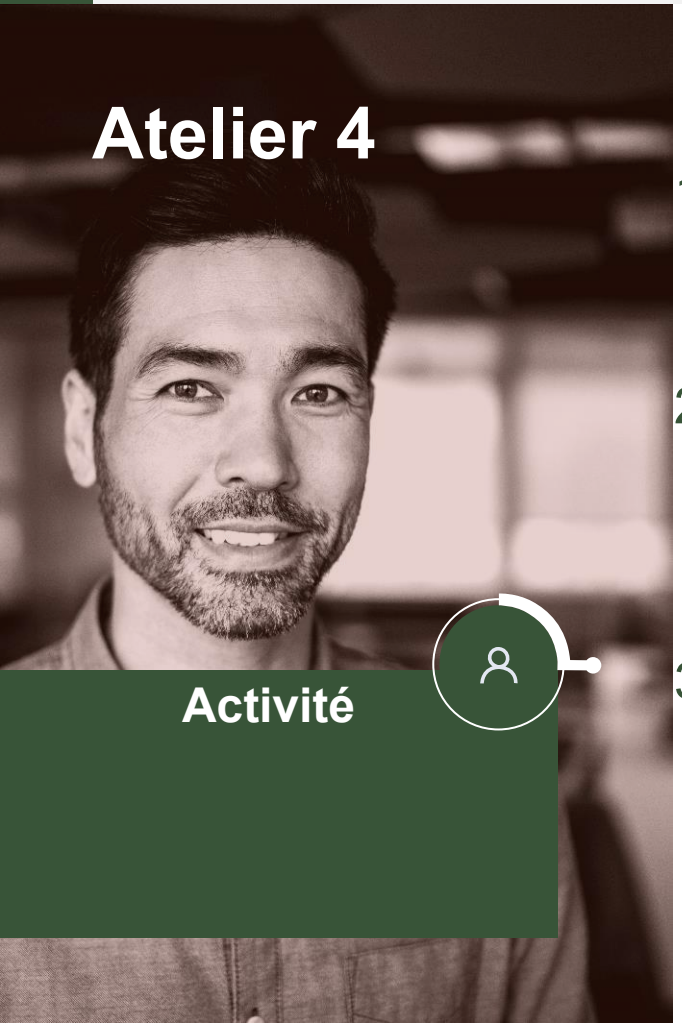
# Les opérateurs ensemblistes

L'opérateur NOT IN:

Cet opérateur permet d'effectuer une DIFFERENCE entre les enregistrements sélectionnés par deux clauses SELECT, c'est-à-dire sélectionner les enregistrements de la première table n'appartenant pas à la seconde.

```
SELECT colonne1, colonne2
FROM table1
WHERE (colonne1, colonne2) NOT IN (
    SELECT colonne1, colonne2
    FROM table2
);
```

# Atelier 4



**Activité**

Je reste disponible pour toute question

1. Affichez les sociétés, adresse et villes de résidence pour tous les tiers de l’entreprise. Trier pas société
2. Affichez toutes les commandes qui comportent en même temps des produits de catégorie 1 du fournisseur 1 et des produits de catégorie 2 du fournisseur 2.
3. Affichez la liste des produits que les clients parisiens ne commandent pas.

# Ce qu’il faut retenir

- Les select des requêtes assemblées doivent être synchrones : même nombre de colonnes, même position.

# Module 5
# Sous-interrogations

# Sous-interrogations

– Dans la clause WHERE

# Dans la clause Where

Une caractéristique puissante de SQL est la possibilité qu'un prédicat employé dans une clause WHERE (expression à droite d'un opérateur de comparaison) comporte un SELECT emboîté.

Sous-interrogation à une ligne et une colonne

Dans ce cas, le SELECT imbriqué équivaut à une valeur.

WHERE exp op (SELECT ...)

où op est un des opérateurs : =, !=, <>, <, >, <=, >=
[not] in

# Dans la clause Where

Exemple :

afficher le nom des employés étant dans le même département que Mercier :

```
SELECT NOME FROM EMP
WHERE DEPT = (SELECT DEPT FROM EMP
                    WHERE NOME = 'MERCIER');
```

# Dans la clause Where

## Sous-interrogation ramenant plusieurs lignes

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.

Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

- l'opérateur [NOT] IN
- les opérateurs obtenus en ajoutant ANY ou ALL à la suite des opérateurs de comparaison classique =, <>, <, >, <=, >=
  - ANY : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble (elle est donc fausse si l'ensemble est vide). On remplacera any par la fonction MIN dans la sous-requête.
  - ALL : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble (elle est vraie si l'ensemble est vide).
    On remplacera all par la fonction MAX dans la sous-requête.

# Dans la clause Where

- WHERE exp op ANY (SELECT ...)
- WHERE exp op ALL (SELECT ...)

où op est un des opérateurs =, !=, <>, <, >, <=, >=

- WHERE exp IN (SELECT ...)
- WHERE exp NOT IN (SELECT ...)

Liste des employés gagnant plus que tous les employés du département 30 :

```
SELECT NOME, SAL FROM EMP
WHERE SAL > ALL (SELECT SAL FROM EMP
                 WHERE DEPT=30);
```

Ou :

```
SELECT NOME, SAL FROM EMP
WHERE SAL > (SELECT MAX(SAL) FROM EMP
             WHERE DEPT=30);
```

# Sous-interrogations

– Dans la clause FROM

# Dans la clause From

Syntaxe :

```
Select A.x , A.y, B.z

from table A

Join  ( select x, z  from table ) B on A.x = B.x;
```

Ou avec la jointure dans la clause where  :

```
Select A.x , A.y, B.z

from table A , ( select x, z  from table ) B

Where A.x = B.x;
```

# Dans la clause From

Exemple : part du salaire de chaque employé par rapport à la masse salariale.

```
Select nom, salaire, round(salaire/masse*100)
from employes, (select sum(salaire) masse from employes) b;
```

Ou avec la fonction over() :

```
Select nom, salaire, salaire/sum(salaire) over()*100 from employes;
```

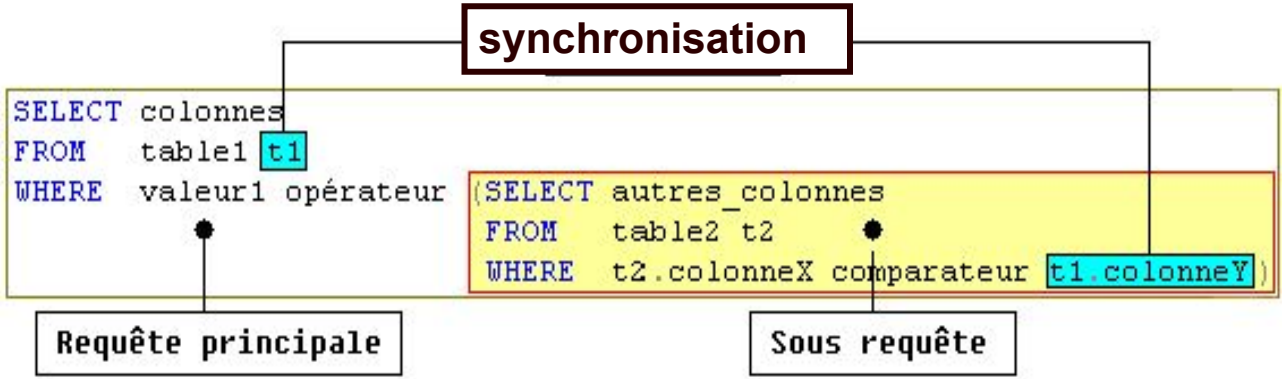
# Sous-interrogations

– Sous-interrogations synchronisées

# Sous-requête synchronisée

- Une sous requête synchronisée est une sous requête qui s'exécute pour chaque ligne de la requête principale et non une fois pour toute.
- Pour arriver à ce résultat, il faut placer une condition dans la sous requête. Condition qui indique que la valeur d'une ou plusieurs colonnes de la sous-requête doit correspondre à la valeur d'une ou plusieurs colonnes de la requête principale.

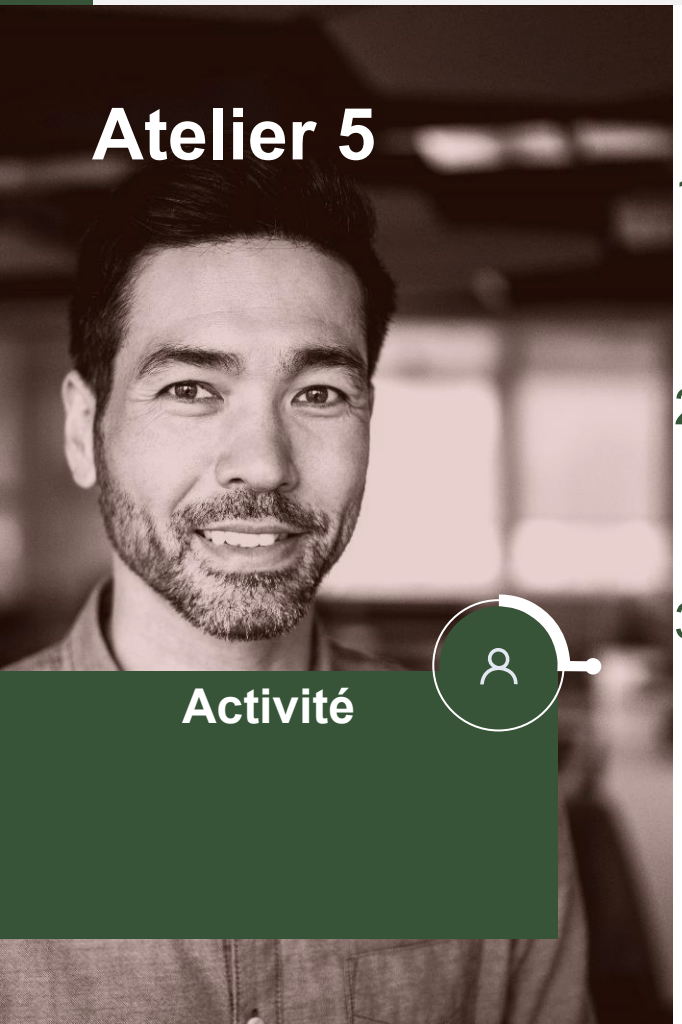
# Sous-requête synchronisée



# Evaluation en ligne

Je vous envoie le lien vers l’ évaluation dans le Tchat de cette réunion Teams

# Atelier 5



Activité

Je reste disponible pour toute question

1. Affichez tous les produits pour lesquels la quantité en stock est inférieure à la moyenne des quantités en stock.
2. Affichez toutes les commandes pour lesquelles les frais de ports dépassent la moyenne des frais de ports pour ce client.
3. Affichez les produits pour lesquels la quantité en stock est supérieure à la quantité en stock de tous les produits de catégorie 3.

# Ce qu’il faut retenir

Echanges

- Une sous-requête dans le where permet d'obtenir ou de calculer une information que l'on ne peux pas connaître.
- Un sous-requête dans le from permet de générer une expression que l'on réutilise ensuite dans la requête principale.

# Module 6
# Modifier les données

# LMD

Le langage de manipulation de données (LMD) est le langage permettant de modifier les informations contenues dans la base.

Il existe trois commandes SQL permettant d'effectuer les trois types de modification des données :

- INSERT = ajout de lignes
- UPDATE = mise à jour de lignes
- DELETE = suppression de lignes

# Insertion

INSERT INTO table (col1,..., coln )

VALUES (val1,...,valn )

ou

INSERT INTO table (col1,..., coln )

SELECT ...

table est le nom de la table sur laquelle porte l'insertion. col1,..., coln est la liste des noms des colonnes pour lesquelles on donne une valeur. Cette liste est optionnelle. Si elle est omise, le SGBD prendra par défaut l'ensemble des colonnes de la table dans l'ordre où elles ont été données lors de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

# Insertion

```
INSERT INTO dept
VALUES (10, 'FINANCES', 'PARIS');

INSERT INTO dept (lieu, nomd, dept)
VALUES ('GRENOBLE', 'RECHERCHE', 20);

INSERT INTO PARTICIPATION (MATR, CODEP)
SELECT MATR, 10 FROM EMP
WHERE NOME = 'MARTIN';
```

# Modification

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table.

```
UPDATE table
SET col1 = exp1, col2 = exp2, ...
WHERE prédicat;
```

ou

```
UPDATE table
SET (col1, col2,...) = (SELECT ...)
WHERE prédicat;
```

table est le nom de la table mise à jour ; col1, col2, ... sont les noms des colonnes qui seront modifiées ; exp1, exp2,... sont des expressions.
On peut faire appel à une sous-interrogation qui nous retourne les valeurs à attribuer aux colonnes (deuxième variante de la syntaxe).

Les valeurs de col1, col2... sont mises à jour dans toutes les lignes satisfaisant le prédicat. La clause WHERE est facultative. Si elle est absente, toutes les lignes sont mises à jour.

# Modification

Faire passer MARTIN dans le département 10 :

```
UPDATE EMP
SET DEPT = 10
WHERE NOME = 'MARTIN‘;
```

Donner à CLEMENT un salaire 10 % au dessus de la moyenne des salaires des secrétaires :

```
UPDATE EMP
SET SAL =  (SELECT AVG(SAL) * 1.10
            FROM EMP
            WHERE POSTE = 'SECRETAIRE')
WHERE NOME = 'CLEMENT‘;
```

# Suppression

L'ordre DELETE permet de supprimer des lignes d'une table.

DELETE FROM table

WHERE prédicat;

La clause WHERE indique quelles lignes doivent être supprimées.

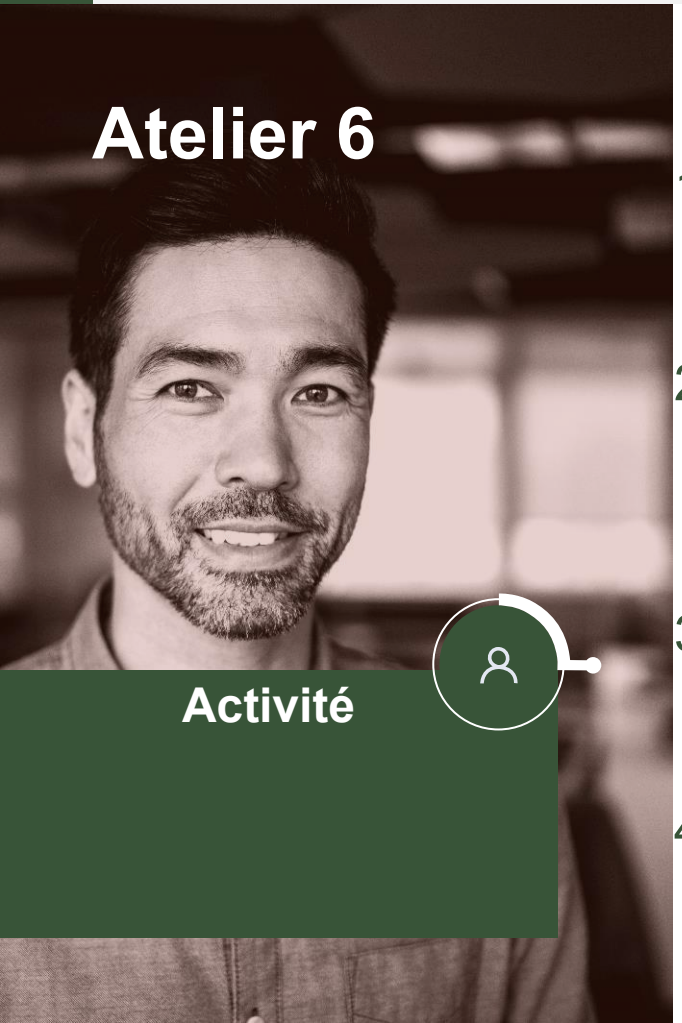
ATTENTION : cette clause est facultative ; si elle n'est pas précisée, TOUTES LES LIGNES DE LA TABLE SONT SUPPRIMEES

# Suppression

```
DELETE FROM dept
WHERE dept = 10;
```

DELETE FROM dept; <= j’efface toutes les lignes

# Atelier 6



**Activité**

Je reste disponible pour toute question

1. Insérez une nouvelle catégorie de produits nommée «fruits et légumes », en respectant les contraintes.
2. Créez un nouveau fournisseur « Grandma » (no_fournisseur = 30) avec les mêmes coordonnées que le fournisseur « Grandma Kelly’s Homestead ».
3. Attribuer les produits de « Grandma Kelly’s Homestead » au nouveau fournisseur précédemment créé.
4. Supprimez l’ancien fournisseur « Grandma Kelly’s Homestead » .

# Module 7
# Notions sur le Langage de Définition de données (LDD)

# Types de données

| TYPE TEXTE |
| --- |
| CHAR |
| VARCHAR |
| LONG |
| … |
| **TYPE NUMERIQUE** |
| NUMBER |
| INTEGER |
| FLOAT |
| BLOB, BIT |
| INTEGER |
| … |
| **TYPE DATE** |
| DATE |
| … |

# Création de table

CREATE TABLE nom_table
(colonne1 type1 contraintes,
colonne2 type2 contraintes,
.....
.....
, contraintes de table)
;

table est le nom que l'on donne à la table ;

colonne1, colonne2,.. sont les noms des colonnes ; type1, type2,.. sont les types des données qui seront contenues dans les colonnes.

# Contraintes

- Contraintes de colonne :
  [NOT] NULL | UNIQUE | PRIMARY KEY |
  CHECK ( prédicat_de_colonne ) |
  FOREIGN KEY [colonne] REFERENCES table (colonne)

- Contraintes de table :
  CONSTRAINT nom_contrainte  UNIQUE |
  PRIMARY KEY ( liste_colonne ) |
  DEFAULT valeur |
  CHECK ( prédicat_de_table ) |
  FOREIGN KEY liste_colonne REFERENCES nom_table (liste_colonne)

# Contraintes

Une colonne peut donc recevoir les contraintes suivantes :

NULL / NOT NULL : précise si la valeur est obligatoire.

DEFAULT : valeur par défaut qui est placée dans la colonne lors des insertions et de certaines opération particulières, lorsque l'on a pas donné de valeur explicite à la colonne.

PRIMARY KEY : précise si la colonne est la clef primaire de la table.

UNIQUE : les valeurs de la colonne doivent être uniques (pas de doublon)

CHECK : permet de préciser un prédicat qui acceptera la valeur s'il est vérifié

FOREIGN KEY : permet, pour les valeurs de la colonne, de faire référence à des valeurs existantes dans une colonne d'une autre table.

# Création de table

Exemple:

```
CREATE TABLE T_CLIENT
(
CLI_ID INTEGER NOT NULL PRIMARY KEY,
CLI_NOM CHAR(32) NOT NULL,
CLI_PRENOM VARCHAR(32)
);
```

# Création de table

Exemple:

```
CREATE TABLE T_VOITURE
(
VTR_ID INTEGER NOT NULL PRIMARY KEY,
VTR_MARQUE CHAR(32) NOT NULL,
VTR_MODELE VARCHAR(16),
VTR_IMMATRICULATION CHAR(10) NOT NULL UNIQUE,
VTR_COULEUR CHAR(5) CHECK (VTR_COULEUR IN ('BLANC', 'NOIR', 'ROUGE', 'VERT', 'BLEU'))
);
```

# Modifier une table

ALTER TABLE nom_table

Ajout d'une colonne – ADD :

ALTER TABLE table ADD col1 type1…;

Modification d'une colonne – Oracle et MySQL :

ALTER TABLE table MODIFY col1 type1, ...;

Sql Server et Postgre :

ALTER TABLE table ALTER COLUMN col1 type1, ...;

Suppression d'une colonne - DROP COLUMN

ALTER TABLE table DROP COLUMN col;

La colonne supprimée ne doit pas être référencée par une clé étrangère

# Modifier une table

```
ALTER TABLE T_CLIENT
ADD CLI_PRENOM VARCHAR(25);

ALTER TABLE T_CLIENT
ADD CLI_DATE_NAISSANCE DATE;

ALTER TABLE T_CLIENT ADD CONSTRAINT
CHK_DATE_NAISSANCE CHECK (CLI_DATE_NAISSANCE
BETWEEN '1880-01-01' AND '2020-01-01');
```

# Modifier une table

Ajouter ou supprimer une contrainte :

Des contraintes d'intégrité peuvent être ajoutées ou supprimées par la commande ALTER TABLE. On ne peut ajouter que des contraintes de table.

```
ALTER TABLE EMP DROP CONSTRAINT NOM_UNIQUE;
```

```
ALTER TABLE EMP ADD CONSTRAINT SAL_MIN CHECK(SAL +
NVL(COMM,0) > 1000);
```

# Supprimer une table

```
DROP TABLE nom_objet;
```

```
DROP TABLE TMP_IMP_DATE;
```

# Les vues

Une vue est une vision des données d'une ou plusieurs tables de la base.
La définition d'une vue est donnée par un SELECT qui indique les données de la base qui seront vues.

La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue :
CREATE VIEW vue (col1, col2...) AS SELECT ...;

Exemple:
CREATE VIEW EMP10 AS SELECT * FROM EMP
WHERE DEPT = 10;

# Les index

Un index permet d'améliorer le temps d'exécution des requêtes d'interrogation de données

CREATE INDEX :

CREATE [UNIQUE] INDEX nom_index ON table (col1, col2,...);

DROP INDEX :

DROP INDEX nom_index [ON table];

# Atelier 7 Facultatif

**Activité**

Je reste disponible pour toute question

1. Créer une table pays avec 2 champs : code pays (4 caractères, clé primaire), nom pays (40 caractères maximum)
2. Ajouter une colonne courriel (60 caractères possibles) à la table CLIENTS. Puis la modifier pour passer à 75 caractères. Pour finir, supprimer cette colonne.
3. Créez une vue qui affiche le nom de la société, l'adresse, le téléphone et la ville des clients qui habitent en province.


Echanges

- Vos attentes
- Vos remarques sur cette formation

# Questions / réponses

- Revenons sur les questions hors plan de cours que vous m’avez posé durant la formation pour y répondre

# Votre formateur

Clément Hamon

clement.hamon35@gmail.com
https://www.linkedin.com/in/clément-hamon-135485209/

# Bilan formation et remerciements

- Merci d’avoir participé à cette formation M2I.
- Envoie du Bilan formation.

# Merci d’avoir suivi cette formation ib Cegos

## et à bientôt !