

Bases de données

Informatique tronc commun

Sylvain Pelletier

PSI - LMSC

Le problème de la gestion de l'information

- ▶ L'informatique est la science de l'information et de son traitement. On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont hétérogènes,
 - ▶ on veut y accéder rapidement, de différentes manières,
 - ▶ avec un système multi-utilisateur sécurisé,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont hétérogènes,
 - ▶ on veut y accéder rapidement, de différentes manières,
 - ▶ avec un système multi-utilisateur sécurisé,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont hétérogènes,
 - ▶ on veut y accéder rapidement, de différentes manières,
 - ▶ avec un système multi-utilisateur sécurisé,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont **hétérogènes**,
 - ▶ on veut y accéder rapidement, **de différentes manières**,
 - ▶ avec un système **multi-utilisateur sécurisé**,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont **hétérogènes**,
 - ▶ on veut y accéder rapidement, **de différentes manières**,
 - ▶ avec un système **multi-utilisateur sécurisé**,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont **hétérogènes**,
 - ▶ on veut y accéder rapidement, **de différentes manières**,
 - ▶ avec un système **multi-utilisateur sécurisé**,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont **hétérogènes**,
 - ▶ on veut y accéder rapidement, **de différentes manières**,
 - ▶ avec un système **multi-utilisateur sécurisé**,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

Le problème de la gestion de l'information

- ▶ **L'informatique est la science de l'information et de son traitement.** On souhaite traiter un grand nombre de données.
 - ▶ Exemple : gestion de la réservation des places dans les trains, gestion des ordres bancaires, Parcoursup...
-
- ▶ On a beaucoup de données, ces données sont **hétérogènes**,
 - ▶ on veut y accéder rapidement, **de différentes manières**,
 - ▶ avec un système **multi-utilisateur sécurisé**,
 - ▶ et un risque d'écriture simultanées, une gestion des pannes, etc.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a redondance de l'information,
- le risque d'erreur est plus fort,
- difficile de faire des recherches selon plusieurs critères.

► Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.

- Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.
- Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

- Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.
- Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.
- Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

- Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.
- Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.
- Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

► Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.

► Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.

► Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

► Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.

► Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.

► Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

► Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.

- Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.
- Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

► Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.

► Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.

► Vous pouvez tester et modifier cette base de données en ligne.

► Stocker l'information dans des fichiers tableurs n'est pas efficace :

- il y a **redondance** de l'information,
- **le risque d'erreur** est plus fort,
- difficile **de faire des recherches** selon plusieurs critères.

- Pour stocker efficacement l'information, on utilise des **tables de données** regroupées en des **base de données**.
- Dans ce cours, on va se baser sur les bases de données du site **w3school** disponible en ligne. Il s'agit d'une base de données d'un magasin qui vend des plats cuisinés.
- Vous pouvez tester et modifier cette base de données en ligne.

- ▶ L'information est stockées dans 8 tables :
 - **Customers**: les clients (91)
 - **Products**: les produits (77)
 - **Orders**: les commandes (196)
 - **OrderDetails**: les détails sur les commandes (518)
 - **Employees**: les employés (10)
 - **Categories**: les catégories de plats (8)
 - **Shippers**: les expéditeurs (3)
 - **Suppliers**: les fournisseurs (29)

Tables Orders, Customers, Employees

Orders	Customers	Employees
OrderID	CustomerID	EmployeeID
CustomerID	CustomerName	LastName
EmployeeID	ContactName	FirstName
OrderDate	Address	BirthDate
ShipperID	City	Photo
	PostalCode	Notes
	Country	

On utilise des identifiants (ID) et trois tables différentes.

Table Orders (commandes)

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3

Table Employees

EmployeeID	LastName	FirstName	BirthDate	Photo
5	Buchanan	Steven	3/4/1955	EmplD5.pic
Notes				
Steven Buchanan graduated from St. Andrews University				

Table Customers (clients)

CustomerID	CustomerName	Address	City	PostalCode	Country
90	Wilman Kala	Keskuskatu 45	Helsinki	21240	Finland

L'information est stockée de manière plus efficace

- ▶ Une **table de données** est un tableau stockant de l'information sur un type d'objets.
- ▶ Elle contient des **enregistrements** ou des **lignes** : les caractéristiques d'un objet.
- ▶ Les **attributs** d'un objet sont les caractéristiques d'un objet, c'est donc les **colonnes** de la table.
- ▶ À un attribut est associé un **domaine** c'est l'ensemble des valeurs que peut prendre cet attribut (c'est le type de données).
- ▶ Le **schéma** de la table est le produit cartésien des domaines des attributs.

- ▶ Une **table de données** est un tableau stockant de l'information sur un type d'objets.
- ▶ Elle contient des **enregistrements** ou des **lignes** : les caractéristiques d'un objet.
- ▶ Les **attributs** d'un objet sont les caractéristiques d'un objet, c'est donc les **colonnes** de la table.
- ▶ À un attribut est associé un **domaine** c'est l'ensemble des valeurs que peut prendre cet attribut (c'est le type de données).
- ▶ Le **schéma** de la table est le produit cartésien des domaines des attributs.

- ▶ Une **table de données** est un tableau stockant de l'information sur un type d'objets.
- ▶ Elle contient des **enregistrements** ou des **lignes** : les caractéristiques d'un objet.
- ▶ Les **attributs** d'un objet sont les caractéristiques d'un objet, c'est donc les **colonnes** de la table.
- ▶ À un attribut est associé un **domaine** c'est l'ensemble des valeurs que peut prendre cet attribut (c'est le type de données).
- ▶ Le **schéma** de la table est le produit cartésien des domaines des attributs.

- ▶ Une **table de données** est un tableau stockant de l'information sur un type d'objets.
- ▶ Elle contient des **enregistrements** ou des **lignes** : les caractéristiques d'un objet.
- ▶ Les **attributs** d'un objet sont les caractéristiques d'un objet, c'est donc les **colonnes** de la table.
- ▶ À un attribut est associé un **domaine** c'est l'ensemble des valeurs que peut prendre cet attribut (c'est le type de données).
- ▶ Le **schéma** de la table est le produit cartésien des domaines des attributs.

- ▶ Une **table de données** est un tableau stockant de l'information sur un type d'objets.
- ▶ Elle contient des **enregistrements** ou des **lignes** : les caractéristiques d'un objet.
- ▶ Les **attributs** d'un objet sont les caractéristiques d'un objet, c'est donc les **colonnes** de la table.
- ▶ À un attribut est associé un **domaine** c'est l'ensemble des valeurs que peut prendre cet attribut (c'est le type de données).
- ▶ Le **schéma** de la table est le produit cartésien des domaines des attributs.

Exemple de schéma

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
ContactName (str)	EmployeeID (int)	FirstName (str)
Address (str)	OrderDate (date)	BirthDate (date)
City (str)	ShipperID (int)	Photo (file)
PostalCode (int)		Notes (text)
Country (str)		

int = entier, str = courte chaîne de caractère (=char), text = long texte

- ▶ Des attributs permettent d'identifier chaque enregistrement, on parle de **clés primaires** (ex : CustomerID).
- ▶ Une clé primaire n'est pas forcément associée à un unique attribut (**clés primaires composites**) (ex : (LastName, FirstName, BirthDate)).
- ▶ Une table A peut contenir comme attribut une clé primaire d'une table B (ex : CustomerID clé primaire de Customers, **clé étrangère** dans Orders). **bien repérer ses liens entre tables !!**

Exemple de schéma

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
ContactName (str)	EmployeeID (int)	FirstName (str)
Address (str)	OrderDate (date)	BirthDate (date)
City (str)	ShipperID (int)	Photo (file)
PostalCode (int)		Notes (text)
Country (str)		

int = entier, str = courte chaîne de caractère (=char), text = long texte

- ▶ Des attributs permettent d'identifier chaque enregistrement, on parle de **clés primaires** (ex : CustomerID).
- ▶ Une clé primaire n'est pas forcément associée à un unique attribut (**clés primaires composites**) (ex : (LastName, FirstName, BirthDate)).
- ▶ Une table A peut contenir comme attribut une clé primaire d'une table B (ex : CustomerID clé primaire de Customers, **clé étrangère** dans Orders). **bien repérer ses liens entre tables !!**

Exemple de schéma

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
ContactName (str)	EmployeeID (int)	FirstName (str)
Address (str)	OrderDate (date)	BirthDate (date)
City (str)	ShipperID (int)	Photo (file)
PostalCode (int)		Notes (text)
Country (str)		

int = entier, str = courte chaîne de caractère (=char), text = long texte

- ▶ Des attributs permettent d'identifier chaque enregistrement, on parle de **clés primaires** (ex : CustomerID).
- ▶ Une clé primaire n'est pas forcément associée à un unique attribut (**clés primaires composites**) (ex : (LastName, FirstName, BirthDate)).
- ▶ Une table A peut contenir comme attribut une clé primaire d'une table B (ex : CustomerID clé primaire de Customers, **clé étrangère** dans Orders). **bien repérer ses liens entre tables!!**

Théorie du modèle entité - association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On doit stocker **des entités et des associations entre ces entités**.
- ▶ Dans notre exemple, les entités sont Customers et Employees. Les tables stockent alors **les informations sur ces entités**.
- ▶ Ces entités / objets sont en **association** (les acheteurs font des achats).
- ▶ **Les associations entre les objets sont alors stockés eux même comme des entités** dans une autre table. C'est la table Orders.
- ▶ Ces associations se retrouvent dans les **clés étrangères** : la table "du milieu" Orders contient comme attributs les clés primaires de Customers et de Employees.

Théorie du modèle entité - association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On doit stocker **des entités et des associations entre ces entités**.
- ▶ Dans notre exemple, les entités sont Customers et Employees. Les tables stockent alors **les informations sur ces entités**.
- ▶ Ces entités / objets sont en **association** (les acheteurs font des achats).
- ▶ Les associations entre les objets sont alors stockés eux même comme **des entités** dans une autre table. C'est la table Orders.
- ▶ Ces associations se retrouvent dans les **clés étrangères** : la table "du milieu" Orders contient comme attributs les clés primaires de Customers et de Employees.

Théorie du modèle entité - association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On doit stocker **des entités et des associations entre ces entités**.
- ▶ Dans notre exemple, les entités sont Customers et Employees. Les tables stockent alors **les informations sur ces entités**.
- ▶ Ces entités / objets sont en **association** (les acheteurs font des achats).
- ▶ Les associations entre les objets sont alors stockés eux même comme **des entités** dans une autre table. C'est la table Orders.
- ▶ Ces associations se retrouvent dans les **clés étrangères** : la table "du milieu" Orders contient comme attributs les clés primaires de Customers et de Employees.

Théorie du modèle entité - association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On doit stocker **des entités et des associations entre ces entités**.
- ▶ Dans notre exemple, les entités sont Customers et Employees. Les tables stockent alors **les informations sur ces entités**.
- ▶ Ces entités / objets sont en **association** (les acheteurs font des achats).
- ▶ **Les associations entre les objets sont alors stockés eux même comme des entités** dans une autre table. C'est la table Orders.
- ▶ Ces associations se retrouvent dans les **clés étrangères** : la table "du milieu" Orders contient comme attributs les clés primaires de Customers et de Employees.

Théorie du modèle entité - association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On doit stocker **des entités et des associations entre ces entités**.
- ▶ Dans notre exemple, les entités sont Customers et Employees. Les tables stockent alors **les informations sur ces entités**.
- ▶ Ces entités / objets sont en **association** (les acheteurs font des achats).
- ▶ **Les associations entre les objets sont alors stockés eux même comme des entités** dans une autre table. C'est la table Orders.
- ▶ Ces associations se retrouvent dans les **clés étrangères** : la table "du milieu" Orders contient comme attributs les clés primaires de Customers et de Employees.

Cardinalité d'une association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On parle **cardinalité** d'une association : **combien de fois au maximum** une entité de type 1 est associée à une entité de type 2.
- ▶ **Association Customers et Orders** : un consommateurs fait des commandes.
- ▶ Cette association est du type : $1 - *$ (ou $1 - n$) : un acheteurs peut participer à plusieurs commandes, mais une commande ne correspond qu'à un acheteur.
- ▶ De même l'association , Employees et Orders est du type $1 - *$. Une commande n'est faite que par un employés, mais un employés peut faire plusieurs commandes.

Cardinalité d'une association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On parle **cardinalité** d'une association : **combien de fois au maximum** une entité de type 1 est associée à une entité de type 2.
- ▶ **Association Customers et Orders** : un consommateurs fait des commandes.
- ▶ Cette association est du type : $1 - *$ (ou $1 - n$) : un acheteurs peut participer à plusieurs commandes, mais une commande ne correspond qu'à un acheteur.
- ▶ De même l'association , Employees et Orders est du type $1 - *$. Une commande n'est faite que par un employés, mais un employés peut faire plusieurs commandes.

Cardinalité d'une association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On parle **cardinalité** d'une association : **combien de fois au maximum** une entité de type 1 est associée à une entité de type 2.
- ▶ **Association Customers et Orders** : un consommateurs fait des commandes.
- ▶ Cette association est du type : $1 - *$ (ou $1 - n$) : un acheteurs peut participer à plusieurs commandes, mais une commande ne correspond qu'à un acheteur.
- ▶ De même l'association , Employees et Orders est du type $1 - *$. Une commande n'est faite que par un employés, mais un employés peut faire plusieurs commandes.

Cardinalité d'une association

Customers	Orders	Employees
CustomerID (int)	OrderID (int)	EmployeeID (int)
CustomerName (str)	CustomerID (int)	LastName (str)
	EmployeeID (int)	

- ▶ On parle **cardinalité** d'une association : **combien de fois au maximum** une entité de type 1 est associée à une entité de type 2.
- ▶ **Association Customers et Orders** : un consommateurs fait des commandes.
- ▶ Cette association est du type : $1 - *$ (ou $1 - n$) : un acheteurs peut participer à plusieurs commandes, mais une commande ne correspond qu'à un acheteur.
- ▶ De même l'association , Employees et Orders est du type $1 - *$. Une commande n'est faite que par un employés, mais un employés peut faire plusieurs commandes.

Types d'associations

On distingue trois cas importants :

- ▶ Association 1 – 1 : on stocke les attributs d'entités en deux tables différentes.

La table `table2` contient la clé primaire de `table1` une fois au plus.

- ▶ Association 1 – * : une entité de `table1` peut être en association avec plusieurs entités de `table2`

La table `table2` contient la clé primaire de `table1` comme clé étrangère un nombre quelconque de fois.

Association entre `Customers` et `Orders`.

- ▶ Association * – * : on veut lier plusieurs entités de deux tables.

Il est nécessaire de créer une table “au milieu” qui permet de séparer une association * – * en deux associations 1 – *.

Association entre `Customers` et `Employees`, on utilise la table `Orders`.

On a souvent cette structure en trois tables !

Types d'associations

On distingue trois cas importants :

- ▶ Association 1 – 1 : on stocke les attributs d'entités en deux tables différentes.

La table `table2` contient la clé primaire de `table1` une fois au plus.

- ▶ Association 1 – * : une entité de `table1` peut être en association avec plusieurs entités de `table2`

La table `table2` contient la clé primaire de `table1` comme clé étrangère un nombre quelconque de fois.

Association entre `Customers` et `Orders`.

- ▶ Association * – * : on veut lier plusieurs entités de deux tables.

Il est nécessaire de créer une table "au milieu" qui permet de séparer une association * – * en deux associations 1 – *.

Association entre `Customers` et `Employees`, on utilise la table `Orders`.

On a souvent cette structure en trois tables !

Types d'associations

On distingue trois cas importants :

- ▶ Association 1 – 1 : on stocke les attributs d'entités en deux tables différentes.

La table `table2` contient la clé primaire de `table1` une fois au plus.

- ▶ Association 1 – * : une entité de `table1` peut être en association avec plusieurs entités de `table2`

La table `table2` contient la clé primaire de `table1` comme clé étrangère un nombre quelconque de fois.

Association entre `Customers` et `Orders`.

- ▶ Association * – * : on veut lier plusieurs entités de deux tables.

Il est nécessaire de créer une table “au milieu” qui permet de séparer une association * – * en deux associations 1 – *.

Association entre `Customers` et `Employees`, on utilise la table `Orders`.

On a souvent cette structure en trois tables !

Types d'associations

On distingue trois cas importants :

- ▶ Association 1 – 1 : on stocke les attributs d'entités en deux tables différentes.

La table `table2` contient la clé primaire de `table1` une fois au plus.

- ▶ Association 1 – * : une entité de `table1` peut être en association avec plusieurs entités de `table2`

La table `table2` contient la clé primaire de `table1` comme clé étrangère un nombre quelconque de fois.

Association entre `Customers` et `Orders`.

- ▶ Association * – * : on veut lier plusieurs entités de deux tables.

Il est nécessaire de créer une table “au milieu” qui permet de séparer une association * – * en deux associations 1 – *.

Association entre `Customers` et `Employees`, on utilise la table `Orders`.

On a souvent cette structure en trois tables !

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

Vocabulaire des bases de données

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

- ▶ **table ou relation** : structure qui contient les informations sur les entités ou leurs relations.
- ▶ **attribut ou colonne** : dans une table, caractéristiques d'un objets.
- ▶ **domaine d'un attribut** : ensemble de définition d'un attribut, type de données d'une colonne.
- ▶ **schéma d'une table** : produit cartésien des domaines. Indique quel est la structure de la table.
- ▶ **enregistrement ou ligne** : information sur un objet donné.
- ▶ **clé primaire** : un ou plusieurs attributs qui caractérisent chaque objet. On parle aussi de **clé primaire composite** lorsque plusieurs attributs caractérisent chaque objet.
- ▶ **clé étrangère** : attribut qui est la clé primaire d'une autre table.
- ▶ **cardinalité d'une association** : nombre de fois qu'une clé étrangère peut être présente (au maximum).

- ▶ Concevoir des bases de données est un métier.
- ▶ On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations.
- ▶ Il s'agit d'apprendre la syntaxe du langage de requête MySql et de l'appliquer au problème posé.
- ▶ Souvent quelques questions de MySql qu'il est intéressant de traiter en premier.

- ▶ Concevoir des bases de données est un métier.
- ▶ On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations.
- ▶ Il s'agit d'apprendre la syntaxe du langage de requête MySql et de l'appliquer au problème posé.
- ▶ Souvent quelques questions de MySql qu'il est intéressant de traiter en premier.

- ▶ Concevoir des bases de données est un métier.
- ▶ On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations.
- ▶ Il s'agit d'apprendre la syntaxe du langage de requête MySql et de l'appliquer au problème posé.
- ▶ Souvent quelques questions de MySql qu'il est intéressant de traiter en premier.

- ▶ Concevoir des bases de données est un métier.
- ▶ On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations.
- ▶ Il s'agit d'apprendre la syntaxe du langage de requête MySql et de l'appliquer au problème posé.
- ▶ Souvent quelques questions de MySql qu'il est intéressant de traiter en premier.

Projection par SELECT

— tous les attributs

```
SELECT * FROM Table
```

— uniquement att1 et att2

```
SELECT att1, att2 FROM Table
```

— les valeurs distinctes de att1

```
SELECT DISTINCT att1 FROM Table
```

— classés en fonction de att3 ordre croissant (ASC)

```
SELECT att1, att2 FROM Table ORDER BY att3
```

— idem en décroissant

```
SELECT att1, att2 FROM Table ORDER BY att3 DESC
```

— les 5 premiers

```
SELECT att1, att2 FROM Table ORDER BY att3 LIMIT 5
```

— éléments 11 à 30

```
SELECT att1, att2 FROM Table ORDER BY att3  
LIMIT 20 OFFSET 10
```

Exemple d'utilisation de SELECT

Sélection de tous les attributs, de certains attributs, des valeurs distinctes d'un attribut :

```
SELECT * FROM Customers;  
SELECT CustomerName, City, Country FROM Customers;  
SELECT DISTINCT Country FROM Customers;
```

Classement

```
SELECT CustomerName, City, Country FROM Customers  
ORDER BY CustomerName;  
  
SELECT * FROM Products ORDER BY Price DESC;  
  
SELECT * FROM Customers ORDER BY Country ASC,  
CustomerName DESC;
```

SELECT avec LIMIT et OFFSET

Les 6 premiers :

```
SELECT CustomerName, City, Country FROM Customers  
ORDER BY CustomerName LIMIT 6;
```

Du 3ème au 5ème :

```
SELECT CustomerName, City, Country FROM Customers  
ORDER BY CustomerName LIMIT 3 OFFSET 2;
```

Le plus cher :

```
SELECT * FROM Products ORDER BY Price DESC LIMIT 1;
```

Sélection par WHERE

```
SELECT * FROM Table WHERE condition ;
```

- ▶ Les conditions possibles sont constituées avec +, -, *, /, =, <>, <, <=, >, >=, AND, OR, NOT.
- ▶ Étant donné leur utilisation massive, on utilise un format spécial pour les dates et les heures qui permet d'effectuer rapidement des comparaisons. De même, on peut très rapidement comparer des chaînes de caractères.

Tout ceci est hors-programme.

Sélection par WHERE

```
SELECT * FROM Table WHERE condition ;
```

- ▶ Les conditions possibles sont constituées avec +, -, *, /, =, <>, <, <=, >, >=, AND, OR, NOT.
- ▶ Étant donné leur utilisation massive, on utilise un format spécial pour les dates et les heures qui permet d'effectuer rapidement des comparaisons. De même, on peut très rapidement comparer des chaînes de caractères.

Tout ceci est hors-programme.

Exemple d'utilisation de SELECT avec WHERE

```
SELECT * FROM Customers WHERE Country='Mexico';
```

```
SELECT * FROM Orders WHERE OrderDate > '2025';
```

```
SELECT * FROM Orders  
WHERE CustomerID = 10 AND EmployeeID = 23;
```

- ▶ Le **renommage** consiste à créer un alias pour un attribut ou une table.
- ▶ Le mot clé pour le renommage s'appelle AS suivi par le nouveau nom de l'attribut.

- ▶ Le **renommage** consiste à créer un alias pour un attribut ou une table.
- ▶ Le mot clé pour le renommage s'appelle AS suivi par le nouveau nom de l'attribut.

- ▶ Le **renommage** consiste à créer un alias pour un attribut ou une table.
- ▶ Le mot clé pour le renommage s'appelle AS suivi par le nouveau nom de l'attribut.

- ▶ Le **renommage** consiste à créer un alias pour un attribut ou une table.
- ▶ Le mot clé pour le renommage s'appelle **AS** suivi par le nouveau nom de l'attribut.

```
SELECT oldName1 AS newName1, oldName2 AS newName2  
FROM nomTable ;
```

```
SELECT att1 , att2  
FROM nomTable AS nvleTable ;
```

- ▶ Le **renommage** consiste à créer un alias pour un attribut ou une table.
- ▶ Le mot clé pour le renommage s'appelle **AS** suivi par le nouveau nom de l'attribut.

```
SELECT oldName1 AS newName1, oldName2 AS newName2  
FROM nomTable ;
```

```
SELECT att1 , att2  
FROM nomTable AS nvleTable ;
```

Pour l'instant peu d'intérêt mais dans la suite on verra mieux pourquoi il faut introduire des alias.

Exemple d'utilisation de AS

Changer le nom d'une colonne ou d'une table :

```
SELECT CustomerID AS ID FROM Customers;  
SELECT * FROM Customers AS Persons;
```

Appliquer une fonction à plusieurs attributs et en faire un nouvel attribut :

```
SELECT CustomerName ,  
Address+ ', ' +PostalCode+ ' ' +City+ ', ' +Country AS Address  
FROM Customers;
```

Requêtes de lecture utilisant deux tables

Soit table1 et table2 deux tables, basées sur le même schéma.

```
SELECT * FROM table1 UNION table2;  
SELECT * FROM table1 INTERSECT table2;  
SELECT * FROM table1 EXCEPT table2;
```

On peut aussi le faire entre des SELECT si ces requêtes renvoient le même schéma :

```
SELECT att1 , att2 FROM table1  
UNION  
SELECT att1 , att2 FROM table2
```

Si un élément est présent dans les table 1 et 2, on peut distinguer UNION ALL qui garde les deux éléments, et UNION DISTINCT (ou simplement UNION) qui n'en garde qu'un.

Exemples UNION

```
SELECT * FROM Customers1 UNION Customers2
```

```
SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;
```

```
SELECT CustomerID FROM Orders  
WHERE OrderDate > '2020' AND OrderDate < '2022'  
EXCEPT  
SELECT CustomerID FROM Orders  
WHERE OrderDate > '2023'
```

Exemple UNION (avancé)

```
SELECT City , Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City , Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

```
SELECT 'Customer' AS Type, ContactName, City , Country
FROM Customers
UNION
SELECT 'Supplier' AS Type, ContactName, City , Country
FROM Suppliers
```

Jointure – théorie

- ▶ Les jointures permettent de mélanger de relations (des tables) et donc de faire des requêtes croisées.
- ▶ La jointure la plus simple (et en pratique très peu utilisée) est le produit cartésien : à partir d'une table `tab1` de n objets notés (x_i) d'une autre table `tab2` de m objets notés (y_j) , on construit une liste de $n \times m$ objets de la forme (x_i, y_j) pour $i \in \llbracket 1, n \rrbracket$ et $j \in \llbracket 1, m \rrbracket$.
- ▶ La syntaxe consiste simplement à écrire les deux tables :

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

- ▶ C'est inutile, SAUF si on ne garde que les éléments qui ont un lien entre eux. En écrivant :

```
SELECT * FROM table1 , table2  
WHERE table1.primaire = table2.etrangere ;
```

- ▶ Les jointures permettent de mélanger de relations (des tables) et donc de faire des **requêtes croisées**.
- ▶ La jointure la plus simple (et en pratique très peu utilisée) est le **produit cartésien** : à partir d'une table `tab1` de n objets notés (x_i) d'une autre table `tab2` de m objets notés (y_j) , on construit une liste de $n \times m$ objets de la forme (x_i, y_j) pour $i \in \llbracket 1, n \rrbracket$ et $j \in \llbracket 1, m \rrbracket$.
- ▶ La syntaxe consiste simplement à écrire les deux tables :

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

- ▶ C'est inutile, SAUF si on ne garde que les éléments qui ont un lien entre eux. En écrivant :

```
SELECT * FROM table1 , table2  
WHERE table1.primaire = table2.etrangere ;
```

- ▶ Les jointures permettent de mélanger de relations (des tables) et donc de faire des **requêtes croisées**.
- ▶ La jointure la plus simple (et en pratique très peu utilisée) est le **produit cartésien** : à partir d'une table `tab1` de n objets notés (x_i) d'une autre table `tab2` de m objets notés (y_j) , on construit une liste de $n \times m$ objets de la forme (x_i, y_j) pour $i \in \llbracket 1, n \rrbracket$ et $j \in \llbracket 1, m \rrbracket$.
- ▶ La syntaxe consiste simplement à écrire les deux tables :

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

- ▶ C'est inutile, SAUF si on ne garde que les éléments qui ont un lien entre eux. En écrivant :

```
SELECT * FROM table1 , table2  
WHERE table1.primaire = table2.etrangere;
```

- ▶ Les jointures permettent de mélanger de relations (des tables) et donc de faire des **requêtes croisées**.
- ▶ La jointure la plus simple (et en pratique très peu utilisée) est le **produit cartésien** : à partir d'une table `tab1` de n objets notés (x_i) d'une autre table `tab2` de m objets notés (y_j) , on construit une liste de $n \times m$ objets de la forme (x_i, y_j) pour $i \in \llbracket 1, n \rrbracket$ et $j \in \llbracket 1, m \rrbracket$.
- ▶ La syntaxe consiste simplement à écrire les deux tables :

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

- ▶ C'est inutile, SAUF si on ne garde que les éléments qui ont un lien entre eux. En écrivant :

```
SELECT * FROM table1 , table2  
WHERE table1.primaire = table2.etrangere;
```

- ▶ Les jointures permettent de mélanger de relations (des tables) et donc de faire des **requêtes croisées**.
- ▶ La jointure la plus simple (et en pratique très peu utilisée) est le **produit cartésien** : à partir d'une table `tab1` de n objets notés (x_i) d'une autre table `tab2` de m objets notés (y_j) , on construit une liste de $n \times m$ objets de la forme (x_i, y_j) pour $i \in \llbracket 1, n \rrbracket$ et $j \in \llbracket 1, m \rrbracket$.
- ▶ La syntaxe consiste simplement à écrire les deux tables :

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

- ▶ C'est inutile, SAUF si on ne garde que les éléments qui ont un lien entre eux. En écrivant :

```
SELECT * FROM table1 , table2  
WHERE table1.primaire = table2.etrangere ;
```

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne une **clé étrangère** dans la **table2** qui doit être égale à une **clé primaire** de la **table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne une **clé étrangère** dans la **table2** qui doit être égale à une **clé primaire** de la **table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne une **clé étrangère** dans la **table2** qui doit être égale à une **clé primaire** de la **table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne **une clé étrangère dans la table2 qui doit être égale à une clé primaire de la table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne **une clé étrangère dans la table2 qui doit être égale à une clé primaire de la table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

La jointure symétrique

- ▶ La **jointure symétrique** consiste à faire le produit cartésien de deux tables en ne gardant que les éléments qui ont une caractéristique en commun.
- ▶ En pratique on a une syntaxe pour **joindre les tables** :

```
SELECT attributs FROM table1  
JOIN table2 ON table1.clePrimaire = table2.cleEtrangere
```

- ▶ L'égalité concerne **une clé étrangère dans la table2 qui doit être égale à une clé primaire de la table1**.
- ▶ On **préfixe** généralement les attributs par le nom de la table pour ne pas se mélanger. (pas obligatoire si il n'y a pas d'ambiguïté).
- ▶ L'ordre des tables n'a pas d'importance (dans le programme).

Syntaxe de la jointure symétrique

- Récupérer le lien nom du client, numéro de commande, date :

```
SELECT Orders.OrderID , Customers.CustomerName ,  
       Orders.OrderDate  
FROM Orders  
JOIN Customers ON Orders.CustomerID=Customers.CustomerID ;
```

- Si on veut le nom du client qui a fait la commande 10248 :

```
SELECT Customers.CustomerName , Orders.OrderId  
FROM Orders  
JOIN Customers  
ON Customers.CustomerID = Orders.CustomerID  
WHERE Orders.OrderId = 10248 ;
```

Syntaxe de la jointure symétrique

- ▶ Récupérer le lien nom du client, numéro de commande, date :

```
SELECT Orders.OrderID , Customers.CustomerName ,  
       Orders.OrderDate  
FROM Orders  
JOIN Customers ON Orders.CustomerID=Customers.CustomerID ;
```

- ▶ Si on veut le nom du client qui a fait la commande 10248 :

```
SELECT Customers.CustomerName , Orders.OrderId  
FROM Orders  
JOIN Customers  
ON Customers.CustomerID = Orders.CustomerID  
WHERE Orders.OrderId = 10248 ;
```

Syntaxe de la jointure symétrique

- ▶ On peut faire plusieurs jointures

```
SELECT Customers.CustomerName AS nomClient ,  
       Employees.LastName AS nomEmployes  
FROM Orders  
JOIN Customers ON Customers.CustomerID = Orders.CustomerID  
JOIN Employees ON Employees.EmployeeID = Orders.EmployeeID
```

- ▶ On met généralement la table “centrale” dans le FROM mais ce n'est pas obligatoire.
- ▶ Autre exemple :

```
SELECT Orders.OrderID , Customers.CustomerName ,  
       Shippers.ShipperName  
FROM Orders  
JOIN Customers ON Orders.CustomerID = Customers.CustomerID  
JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID ;
```

Syntaxe de la jointure symétrique

- ▶ On peut faire plusieurs jointures

```
SELECT Customers.CustomerName AS nomClient ,  
       Employees.LastName AS nomEmployes  
FROM Orders  
JOIN Customers ON Customers.CustomerID = Orders.CustomerID  
JOIN Employees ON Employees.EmployeeID = Orders.EmployeeID
```

- ▶ On met généralement la table “centrale” dans le FROM mais ce n’est pas obligatoire.
- ▶ Autre exemple :

```
SELECT Orders.OrderID , Customers.CustomerName ,  
       Shippers.ShipperName  
FROM Orders  
JOIN Customers ON Orders.CustomerID = Customers.CustomerID  
JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID ;
```

Syntaxe de la jointure symétrique

- ▶ On peut faire plusieurs jointures

```
SELECT Customers.CustomerName AS nomClient ,  
       Employees.LastName AS nomEmployes  
FROM Orders  
JOIN Customers ON Customers.CustomerID = Orders.CustomerID  
JOIN Employees ON Employees.EmployeeID = Orders.EmployeeID
```

- ▶ On met généralement la table “centrale” dans le FROM mais ce n’est pas obligatoire.
- ▶ Autre exemple :

```
SELECT Orders.OrderID , Customers.CustomerName ,  
       Shippers.ShipperName  
FROM Orders  
JOIN Customers ON Orders.CustomerID = Customers.CustomerID  
JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID ;
```

Fonctions d'agrégations

- ▶ Une fonction d'agrégation est une fonction qui s'applique sur un “agrégats” / “tas” de valeurs : **une liste de longueur quelconque et sans ordre.**
- ▶ Les fonctions d'agrégation à connaître sont : `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(*)`, `COUNT(DISTINCT e)`.
- ▶ Cela permet de faire des **statistiques** sur les données.
- ▶ Sans le `GROUP BY`, on applique à toute la table et on obtient ainsi *une unique valeur*.

— **basique :**

```
SELECT MIN(Price), MAX(Price), AVG(Price) FROM Products;
```

— **avec un where :**

```
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

Fonctions d'agrégations

- ▶ Une fonction d'agrégation est une fonction qui s'applique sur un “agrégats” / “tas” de valeurs : **une liste de longueur quelconque et sans ordre.**
- ▶ Les fonctions d'agrégation à connaître sont : **SUM(e)**, **AVG(e)**, **MAX(e)**, **MIN(e)**, **COUNT(*)**, **COUNT(DISTINCT e)**.
- ▶ Cela permet de faire des **statistiques** sur les données.
- ▶ Sans le GROUP BY, on applique à toute la table et on obtient ainsi *une unique valeur.*

— **basique :**

```
SELECT MIN(Price), MAX(Price), AVG(Price) FROM Products;
```

— **avec un where :**

```
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

Fonctions d'agrégations

- ▶ Une fonction d'agrégation est une fonction qui s'applique sur un “agrégats” / “tas” de valeurs : **une liste de longueur quelconque et sans ordre.**
- ▶ Les fonctions d'agrégation à connaître sont : **SUM(e)**, **AVG(e)**, **MAX(e)**, **MIN(e)**, **COUNT(*)**, **COUNT(DISTINCT e)**.
- ▶ Cela permet de faire des **statistiques** sur les données.
- ▶ Sans le **GROUP BY**, on applique à toute la table et on obtient ainsi *une unique valeur.*

— **basique :**

```
SELECT MIN(Price), MAX(Price), AVG(Price) FROM Products;
```

— **avec un where :**

```
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

Fonctions d'agrégations

- ▶ Une fonction d'agrégation est une fonction qui s'applique sur un “agrégats” / “tas” de valeurs : **une liste de longueur quelconque et sans ordre.**
- ▶ Les fonctions d'agrégation à connaître sont : **SUM(e)**, **AVG(e)**, **MAX(e)**, **MIN(e)**, **COUNT(*)**, **COUNT(DISTINCT e)**.
- ▶ Cela permet de faire des **statistiques** sur les données.
- ▶ Sans le GROUP BY, on applique à toute la table et on obtient ainsi *une unique valeur.*

— **basique :**

```
SELECT MIN(Price), MAX(Price), AVG(Price) FROM Products;
```

— **avec un where :**

```
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

Fonctions d'agrégations

- ▶ Une fonction d'agrégation est une fonction qui s'applique sur un “agrégats” / “tas” de valeurs : **une liste de longueur quelconque et sans ordre.**
- ▶ Les fonctions d'agrégation à connaître sont : **SUM(e)**, **AVG(e)**, **MAX(e)**, **MIN(e)**, **COUNT(*)**, **COUNT(DISTINCT e)**.
- ▶ Cela permet de faire des **statistiques** sur les données.
- ▶ Sans le GROUP BY, on applique à toute la table et on obtient ainsi *une unique valeur.*

— **basique :**

```
SELECT MIN(Price), MAX(Price), AVG(Price) FROM Products;
```

— **avec un where :**

```
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

Syntaxe

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Avec une jointure, combien de tarte au sucre vendues ?

```
SELECT SUM(OrderDetails.Quantity)
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
WHERE Products.ProductName = 'Tarte au sucre';
```

Syntaxe

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Avec une jointure, combien de tarte au sucre vendues ?

```
SELECT SUM(OrderDetails.Quantity)
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
WHERE Products.ProductName = 'Tarte au sucre';
```

- ▶ Souvent, on veut faire regrouper les valeurs d'une table selon un ou plusieurs attributs. On fait un **tas / agrégat** des entités qui ont un attribut en commun.
- ▶ La machine regroupe d'abord les valeurs qui ont cet attribut (ou ces attributs) en commun, puis on peut calculer des fonctions d'agrégation sur ce groupe, ou récupérer un attribut commun à ce groupe.
- ▶ Exemple de syntaxe :

```
SELECT COUNT(*) FROM Products GROUP BY CategoryID;
```

- ▶ Souvent, on veut faire regrouper les valeurs d'une table selon un ou plusieurs attributs. On fait un **tas / agrégat** des entités qui ont un attribut en commun.
- ▶ La machine regroupe d'abord les valeurs qui ont cet attribut (ou ces attributs) en commun, puis on peut calculer des fonctions d'agrégation sur ce groupe, ou récupérer un attribut commun à ce groupe.
- ▶ Exemple de syntaxe :

```
SELECT COUNT(*) FROM Products GROUP BY CategoryID;
```

- ▶ Souvent, on veut faire regrouper les valeurs d'une table selon un ou plusieurs attributs. On fait un **tas / agrégat** des entités qui ont un attribut en commun.
- ▶ La machine regroupe d'abord les valeurs qui ont cet attribut (ou ces attributs) en commun, puis on peut calculer des fonctions d'agrégation sur ce groupe, ou récupérer un attribut commun à ce groupe.
- ▶ Exemple de syntaxe :

```
SELECT COUNT(*) FROM Products GROUP BY CategoryID ;
```

Agrégation

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Pour chaque nom de produit, combien de commandes ont été faites ?

```
SELECT SUM(OrderDetails.Quantity), Products.ProductName
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID
```

Agrégation

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Pour chaque nom de produit, combien de commandes ont été faites ?

```
SELECT SUM(OrderDetails.Quantity), Products.ProductName
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID
```

Agrégation

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Pour chaque numéro de commande, combien s'élève le total ?

```
SELECT SUM(Price * Quantity) AS total , OrderID AS numCommande
FROM OrderDetails
JOIN Products ON OrderDetails.ProductID = Products.ProductID
GROUP BY OrderDetails.OrderID
```

Agrégation

Products	OrderDetails
ProductID	OrderDetailID
ProductName	OrderID
SupplierID	ProductID
CategoryID	Quantity
Unit	
Price	

Pour chaque numéro de commande, combien s'élève le total ?

```
SELECT SUM(Price * Quantity) AS total , OrderID AS numCommande
FROM OrderDetails
JOIN Products ON OrderDetails.ProductID = Products.ProductID
GROUP BY OrderDetails.OrderID
```

Agrégation

Orders	Shippers
OrderID	ShipperID
CustomerID	ShipperName
EmployeeID	Phone
OrderDate	
ShipperID	

Pour chaque Shippers, combien de commandes?

```
SELECT Shippers.ShipperName, COUNT(*) AS NumberOfOrders
FROM Orders
JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Agrégation

Orders	Shippers
OrderID	ShipperID
CustomerID	ShipperName
EmployeeID	Phone
OrderDate	
ShipperID	

Pour chaque Shippers, combien de commandes ?

```
SELECT Shippers.ShipperName, COUNT(*) AS NumberOfOrders
FROM Orders
JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Filtrage des agrégats

- ▶ On peut choisir de ne sélectionner que les agrégats qui vérifient une condition.
- ▶ Attention: cette sélection est faite après l'agrégation. C'est un filtrage des agrégats et non des données.
- ▶ On peut faire dépendre cette condition du résultat d'une fonction d'agrégation.

La syntaxe la plus courante est donc :

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

Filtrage des agrégats

- ▶ On peut choisir de ne sélectionner que les agrégats qui vérifient une condition.
- ▶ Attention: cette sélection est faite après l'agrégation. C'est un filtrage des agrégats et non des données.
- ▶ On peut faire dépendre cette condition du résultat d'une fonction d'agrégation.

La syntaxe la plus courante est donc :

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

Filtrage des agrégats

- ▶ On peut choisir de ne sélectionner que les agrégats qui vérifient une condition.
- ▶ Attention: cette sélection est faite après l'agrégation. C'est un filtrage des agrégats et non des données.
- ▶ On peut faire dépendre cette condition du résultat d'une fonction d'agrégation.

La syntaxe la plus courante est donc :

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

Filtrage des agrégats

- ▶ On peut choisir de ne sélectionner que les agrégats qui vérifient une condition.
- ▶ Attention: cette sélection est faite après l'agrégation. C'est un filtrage des agrégats et non des données.
- ▶ On peut faire dépendre cette condition du résultat d'une fonction d'agrégation.

La syntaxe la plus courante est donc :

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

Filtrage des agrégats

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

- ▶ on regroupe alors les entités selon la valeur de att1,
- ▶ puis on applique la fonction d'agrégation F, ce qui permet d'attribuer une valeur à chacun des groupes.
- ▶ Enfin, on ne garde que les groupes où cette valeur est supérieure à x.

Filtrage des agrégats

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

- ▶ on **regroupe** alors les entités selon la valeur de att1,
- ▶ puis on applique la fonction d'agrégation F, ce qui permet d'attribuer une valeur à chacun des groupes.
- ▶ Enfin, on ne garde que les groupes où cette valeur est supérieure à x.

Filtrage des agrégats

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

- ▶ on **regroupe** alors les entités selon la valeur de att1,
- ▶ puis on applique la fonction d'agrégation F, ce qui permet d'attribuer une valeur à chacun des groupes.
- ▶ Enfin, on ne garde que les groupes où cette valeur est supérieure à x.

```
SELECT att1 , F(att2) AS valeur
FROM table
GROUP BY att1
HAVING valeur >= x
```

- ▶ on **regroupe** alors les entités selon la valeur de att1,
- ▶ puis on applique la fonction d'agrégation F, ce qui permet d'attribuer une valeur à chacun des groupes.
- ▶ Enfin, on ne garde que les groupes où cette valeur est supérieure à x.

Filtrage des agrégats, exemple

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;
```

Donne le nombre d'acheteurs par pays, pour les pays qui ont plus que 5 acheteurs.

Filtrage des agrégats, exemple

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;
```

Donne le nombre d'acheteurs par pays, pour les pays qui ont plus que 5 acheteurs.

Filtrage des agrégats, exemple

```
SELECT COUNT(CustomerID) AS nbrCustomer , Country
FROM Customers
GROUP BY Country
HAVING nbrCustomer > 5
ORDER BY COUNT(CustomerID) DESC;
```

Même chose avec un renommage et par ordre décroissant.

Filtrage des agrégats, exemple

```
SELECT COUNT(CustomerID) AS nbrCustomer , Country
FROM Customers
GROUP BY Country
HAVING nbrCustomer > 5
ORDER BY COUNT(CustomerID) DESC;
```

Même chose avec un renommage et par ordre décroissant.

Le nom des employés et leurs nombres de ventes. En ne gardant que ceux qui ont fait plus de 10 ventes, et en affichant par l'ordre alphabétique de leur nom.

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING    NumberOfOrders > 10;
```

Le nom des employés et leurs nombres de ventes. En ne gardant que ceux qui ont fait plus de 10 ventes, et en affichant par l'ordre alphabétique de leur nom.

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING    NumberOfOrders > 10;
```

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le groupe ou les fonctions d'agrégations.

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le groupe ou les fonctions d'agrégations.

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le groupe ou les fonctions d'agrégations.

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le groupe ou les fonctions d'agrégations.

WHERE OU HAVING ?

```
SELECT Employees.LastName ,  
       COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.OrderDate > '2024'  
GROUP BY Employees.EmployeeID  
HAVING NumberOfOrders > 10;
```

- ▶ WHERE est calculé avant l'agrégation, c'est une sélection sur les ligne / objets.
- ▶ HAVING est calculé après l'agrégation, c'est une sélection / filtrage sur les agrégats.
- ▶ WHERE indique les données que l'on veut utiliser, HAVING permet juste de réduire l'affichage.
- ▶ Dans HAVING on ne peut utiliser que les attributs constants sur le groupe ou les fonctions d'agrégations.

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employés nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second on fait les calculs pour tous les employés mais on ne garde

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employées nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second on fait le calcul pour tous les employés puis on regroupe.

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employés nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second, on fait les calculs pour tous les employés mais on ne garde

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employés nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second, on fait les calculs pour tous les employés mais on ne garde

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employés nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second, on fait les calculs pour tous les employés mais on ne garde

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Employees.BirthDate > '2000'  
GROUP BY Employees.EmployeeID
```

```
SELECT Employees.LastName ,  
        COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
GROUP BY Employees.EmployeeID  
HAVING Employees.BirthDate > '2000'
```

- ▶ Dans les deux cas, on calcule le nombre de vente pour les employés nées après l'an 2000.
- ▶ Dans le premier cas, on sélectionne avant l'agrégation, dans le second, on fait les calculs pour tous les employés mais on ne garde

Séquence complète en SQL

```
SELECT att, f(atts) FROM table1
JOIN table2 ON table1.idx = table2.idx
WHERE condition
GROUP BY att HAVING condition2
ORDER BY f(atts) LIMIT 3 OFFSET 5
```

- 1 Table (FROM)?, croisée avec une autre table (JOIN ... ON ...)?, uniquement les données vérifiant une condition (WHERE)?
- 2 Regroupées selon un ou plusieurs attributs (GROUP BY)??
- 3 Attributs que l'on souhaite récupérer (SELECT)? Fonctions d'agrégations pour obtenir des caractéristiques du groupe?
- 4 afficher uniquement les agrégats vérifiant une condition (HAVING)?
Ordre (ORDER BY? Place des résultats (LIMIT .. OFFSET ..)?

Séquence complète en SQL

```
SELECT att, f(atts) FROM table1
JOIN table2 ON table1.idx = table2.idx
WHERE condition
GROUP BY att HAVING condition2
ORDER BY f(atts) LIMIT 3 OFFSET 5
```

- 1 Table (FROM)?, croisée avec une autre table (JOIN ... ON ...)?, uniquement les données vérifiant une condition (WHERE)?
- 2 Regroupées selon un ou plusieurs attributs (GROUP BY)??
- 3 Attributs que l'on souhaite récupérer (SELECT)? Fonctions d'agrégations pour obtenir des caractéristiques du groupe?
- 4 afficher uniquement les agrégats vérifiant une condition (HAVING)?
Ordre (ORDER BY? Place des résultats (LIMIT .. OFFSET ..)?

Séquence complète en SQL

```
SELECT att , f(att) FROM table1
JOIN table2 ON table1.idx = table2.idx
WHERE condition
GROUP BY att HAVING condition2
ORDER BY f(att) LIMIT 3 OFFSET 5
```

- 1 Table (FROM) ?, croisée avec une autre table (JOIN ... ON ...) ?, uniquement les données vérifiant une condition (WHERE) ?
- 2 Regroupées selon un ou plusieurs attributs (GROUP BY) ??
- 3 Attributs que l'on souhaite récupérer (SELECT) ? Fonctions d'agrégations pour obtenir des caractéristiques du groupe ?
- 4 afficher uniquement les agrégats vérifiant une condition (HAVING) ?
Ordre (ORDER BY ? Place des résultats (LIMIT .. OFFSET ..) ?

Séquence complète en SQL

```
SELECT att, f(atts) FROM table1  
JOIN table2 ON table1.idx = table2.idx  
WHERE condition  
GROUP BY att HAVING condition2  
ORDER BY f(atts) LIMIT 3 OFFSET 5
```

- 1 Table (FROM)?, croisée avec une autre table (JOIN ... ON ...)?, uniquement les données vérifiant une condition (WHERE)?
- 2 Regroupées selon un ou plusieurs attributs (GROUP BY)??
- 3 Attributs que l'on souhaite récupérer (SELECT)? Fonctions d'agrégations pour obtenir des caractéristiques du groupe?
- 4 afficher uniquement les agrégats vérifiant une condition (HAVING)?
Ordre (ORDER BY? Place des résultats (LIMIT .. OFFSET ..)?

Quelques trucs en plus

- ▶ On sélectionne une expression.
- ▶ On peut joindre une table avec elle même
- ▶ On peut joindre sur plusieurs égalités
- ▶ On peut faire des requêtes emboîtées

Quelques trucs en plus

- ▶ On sélectionne une expression.
- ▶ On peut joindre une table avec elle même
- ▶ On peut joindre sur plusieurs égalités
- ▶ On peut faire des requêtes emboîtées

Quelques trucs en plus

- ▶ On sélectionne une expression.
- ▶ On peut joindre une table avec elle même
- ▶ On peut joindre sur plusieurs égalités
- ▶ On peut faire des requêtes emboîtées

Quelques trucs en plus

- ▶ On sélectionne une expression.
- ▶ On peut joindre une table avec elle même
- ▶ On peut joindre sur plusieurs égalités
- ▶ On peut faire des requêtes emboîtées

Table Randonnee

Id entier, identifiant de la randonnée ;

Titre chaîne de caractères, titre de la randonnée ;

Type chaîne de caractères du type de l'activité : "Pied", "VTT", "Cheval" ;

Lieu chaîne de caractères, coordonnées GPS du point de départ ;

Distance flottant, longueur en kilomètres de la randonnée ;

DenP entier, dénivelé positif en mètres ;

DenN entier, dénivelé négatif en mètres ;

Duree entier, durée en minutes de la randonnée ;

Niveau entier compris entre 1 et 5 (1 : facile à 5 : extrême), difficulté ;

IdAuteur entier, identifiant de l'auteur de la randonnée ;

Trace chaîne de caractères, lien internet vers la trace GPX.

Table Auteur

Id entier, identifiant de l'auteur (le randonneur) ;

Nom chaîne de caractères, nom de l'auteur ;

Prenom chaîne de caractères, prénom de l'auteur ;

Pseudo chaîne de caractères, pseudo de l'auteur ;

Mail chaîne de caractères, mail de l'auteur.

Expliquer en quoi l'attribut Titre ne peut probablement pas être une clé primaire pour la table Randonnee. Proposer un attribut de la table Randonnee qui puisse être une clé primaire.

Identifier un attribut qui soit une clé étrangère de la table Randonnee.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

Expliquer en quoi l'attribut `Titre` ne peut probablement pas être une clé primaire pour la table `Randonnee`. Proposer un attribut de la table `Randonnee` qui puisse être une clé primaire.

`Titre` n'est pas unique. `Id` peut être une clé primaire car elle est unique.

Identifier un attribut qui soit une clé étrangère de la table `Randonnee`.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

Expliquer en quoi l'attribut Titre ne peut probablement pas être une clé primaire pour la table Randonnee. Proposer un attribut de la table Randonnee qui puisse être une clé primaire.

Titre n'est pas unique. Id peut être une clé primaire car elle est unique.

Identifier un attribut qui soit une clé étrangère de la table Randonnee.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

Expliquer en quoi l'attribut Titre ne peut probablement pas être une clé primaire pour la table Randonnee. Proposer un attribut de la table Randonnee qui puisse être une clé primaire.

Titre n'est pas unique. Id peut être une clé primaire car elle est unique.

Identifier un attribut qui soit une clé étrangère de la table Randonnee.

IdAuteur est une clé étrangère de la table Randonnee.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

Expliquer en quoi l'attribut Titre ne peut probablement pas être une clé primaire pour la table Randonnee. Proposer un attribut de la table Randonnee qui puisse être une clé primaire.

Titre n'est pas unique. Id peut être une clé primaire car elle est unique.

Identifier un attribut qui soit une clé étrangère de la table Randonnee.

IdAuteur est une clé étrangère de la table Randonnee.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

Exemple CCINP 2025

Expliquer en quoi l'attribut Titre ne peut probablement pas être une clé primaire pour la table Randonnee. Proposer un attribut de la table Randonnee qui puisse être une clé primaire.

Titre n'est pas unique. Id peut être une clé primaire car elle est unique.

Identifier un attribut qui soit une clé étrangère de la table Randonnee.

IdAuteur est une clé étrangère de la table Randonnee.

Écrire une requête SQL dont l'évaluation renvoie le titre, les coordonnées GPS du point de départ et la longueur des randonnées à pied.

```
SELECT Titre , Lieu , Distance
FROM Randonnee WHERE Type = 'Pied'
```

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

```
SELECT IdAuteur , COUNT(*) AS nbActivites
FROM Randonnee
WHERE Type = 'Pied' AND Niveau = 3
GROUP BY IdAuteur
ORDER BY nbActivites DESC
```

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

```
SELECT Auteur.Pseudo , Randonnee.Titre  
FROM Randonnee  
JOIN Auteur ON Auteur.Id = Randonnee.IdAuteur
```

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

Écrire une requête SQL dont l'évaluation renvoie l'identifiant de l'auteur et son nombre d'activités à pied de niveau 3, classées par ordre décroissant du nombre d'activités de chaque auteur.

Écrire une requête SQL dont l'évaluation renvoie le Pseudo de l'auteur et le Titre des randonnées stockées dans la base.

Écrire une requête SQL dont l'évaluation renvoie les nom et prénom d'un des auteurs ayant posté le plus de randonnées à cheval.

```
SELECT Auteur.Nom, Auteur.Prenom
FROM Randonnee
JOIN Auteur ON Auteur.Id = Randonnee.IdAuteur
WHERE Type = 'Cheval'
GROUP BY Randonnee.IdAuteur
ORDER BY COUNT(*) LIMIT 1
```