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...

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.

2/39

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.

2/39

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.

2/39

- ► 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.

3/39

- 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.

3/39

- ► 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.

3/39

- 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.

3/39

- 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.

3/39

- 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.

3/39

- 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.

3/39

- 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.

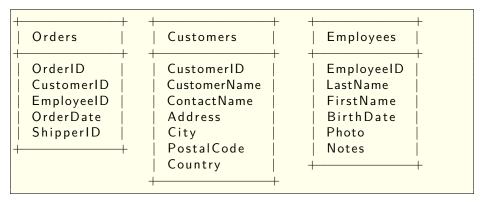
3/39

Structure de la base de données

- 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)

4/39

Tables Orders, Customers, Employees



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

5/39

Table Orders (commandes)

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

Table Employees

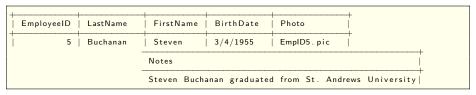


Table Customers (clients)

	I	l			
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.

7/39

- 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.

7/39

- 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.

7/39

- 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.

7/39

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

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

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!!

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

- 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 atributs les clés primaires de Customers et de Employees.

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

- 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 atributs les clés primaires de Customers et de Employees.

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

- 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 atributs les clés primaires de Customers et de Employees.

Customers	Orders Employees
CustomerID (int) CustomerName (str)	OrderID (int) EmployeeID (int) 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 atributs les clés primaires de Customers et de Employees.

Customers	Orders Employees
CustomerID (int) CustomerName (str)	OrderID (int) EmployeeID (int) 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 atributs les clés primaires de Customers et de Employees.

Cardinalité d'une association

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

- ➤ 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) CustomerName (str)	OrderID (int) CustomerID (int) EmployeeID (int)	EmployeeID (int) LastName (str)

- ➤ 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) CustomerName (str)	OrderID (int) CustomerID (int) EmployeeID (int)	EmployeelD (int) LastName (str)

- ➤ 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.

10 / 39

Cardinalité d'une association

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

- ➤ 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.

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 Emplement en utilise le table.

On a souvent cette structure en trois tables

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

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

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!

- ► 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).

Sylvain Pelletier Bases de données PSI - LMSC 12 / 39

- ► 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).

Sylvain Pelletier Bases de données PSI - LMSC 12 / 39

- ► 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).

- ▶ 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).

- ► 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.

13/39

Sylvain Pelletier Bases de données PSI - LMSC

- 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.

13/39

Sylvain Pelletier Bases de données PSI - LMSC

- 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.

Sylvain Pelletier Bases de données PSI - LMSC 13 / 39

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
```

Sylvain Pelletier Bases de données PSI - LMSC 14 / 39

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;
```

Sylvain Pelletier Bases de données PSI - LMSC 15 / 39

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;
```

Sylvain Pelletier Bases de données PSI - LMSC 16 / 39

Sélection par WHERE

SELECT * FROM Table WHERE condition;

- ▶ Les conditions possibles sont constuites 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.

17/39

Sylvain Pelletier Bases de données PSI - LMSC

Sélection par WHERE

SELECT * FROM Table WHERE condition:

- Les conditions possibles sont constuites 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.

17/39

Sylvain Pelletier Bases de données

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;
```

Sylvain Pelletier Bases de données PSI - LMSC 18 / 39

- 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.

Sylvain Pelletier Bases de données PSI - LMSC 19 / 39

- ▶ 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.

Sylvain Pelletier Bases de données PSI - LMSC 19 / 39

- 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.

Sylvain Pelletier Bases de données PSI - LMSC 19 / 39

- 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;
```

Sylvain Pelletier Bases de données PSI - LMSC 19/39

- 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.

Sylvain Pelletier Bases de données PSI - LMSC 19/39

Exemple d'utilsation 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 :

Sylvain Pelletier Bases de données PSI - LMSC 20 / 39

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;
SFLECT CustomerID FROM Orders
WHERE OrderDate > '2020' AND OrderDate < '2022'
FXCFPT
SELECT CustomerID FROM Orders
WHERE OrderDate > '2023'
```

Sylvain Pelletier Bases de données PSI - LMSC 22 / 39

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 $i \in [\![1, n]\!]$ et $j \in [\![1, m]\!]$.
- 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;
```

Sylvain Pelletier Bases de données PSI - LMSC 24/39

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 $i \in [1, n]$ et $j \in [1, m]$.
- 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;
```

10 + 40 + 40 + 40 + 5 + 90 C

Sylvain Pelletier Bases de données PSI - LMSC 24 / 39

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 $i \in [\![1, n]\!]$ et $j \in [\![1, m]\!]$.
- 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;
```

4 □ ▷ ▷ □ ▷ ▷ □ ▷ ○ □ ▷ ○ □ ▷ ○ □ ▷ ○ □

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 $i \in [\![1, n]\!]$ et $j \in [\![1, m]\!]$.
- 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;
```

4 D F 4 B F 4 E F 4 E F 9 Q Q

Sylvain Pelletier Bases de données PSI - LMSC 24/3

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 $i \in [\![1, n]\!]$ et $j \in [\![1, m]\!]$.
- 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 :

- 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 jointer 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 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 jointer 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 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 jointer 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).

25/39

- ► 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 jointer 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).

25/39

- ► 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 jointer 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).

◆ロト ◆問ト ◆意ト ◆意ト ・ 意 ・ 釣り○

25/39

Sylvain Pelletier

- ► 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 jointer 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).

Bases de données

L'ordre des tables n'a pas d'importance (dans le programme).

4 □ ▶ 4 ⓓ ▶ 4 悥 ▶ 4 悥 ▶ □ 틸 · ♡

PSI - LMSC

25/39

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 . Orderld FROM Orders

JOIN Customers

ON Customers . CustomerID = Orders . CustomerID

WHERE Orders . OrderId = 10248;
```

26/39

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;
```

26/39

On peut faire plusieurs jointures

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

On peut faire plusieurs jointures

- 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;
```

PSI - LMSC

On peut faire plusieurs jointures

- 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;
```

- ► 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), MON(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;
```

- ► 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), MON(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;
```

- ► 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), MON(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;
```

- 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), MON(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;

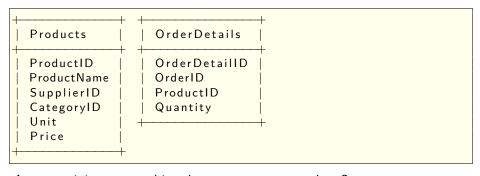
— avec un where:
SELECT COUNT(*) FROM Products WHERE Price > 20;
```

- ▶ 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), MON(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;
```

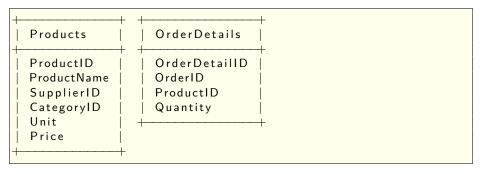
Syntaxe



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



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'abords 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;

30 / 39

- 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'abords 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 :

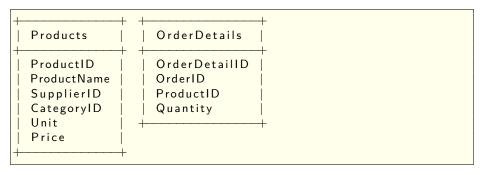
SELECT COUNT(*) FROM Products GROUP BY CategoryID;

30 / 39

- 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'abords 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;

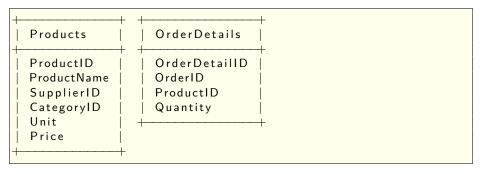
30 / 39



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

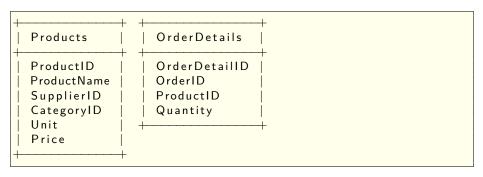
```
SELECT SUM(OrderDetails.Quantity), Products.ProductName
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID
```

বিচাৰ বাচি বিভাগ বাচি



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

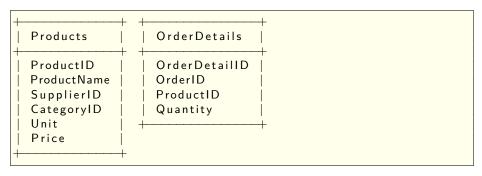
```
SELECT SUM(OrderDetails.Quantity), Products.ProductName
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductID
```



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

```
\begin{array}{lll} \textbf{SELECT SUM} \big( \, \text{Price} \, * \, \, \text{Quantity} \big) \, \, & \textbf{AS total} \, , \, \, \text{OrderID AS numCommande} \\ \textbf{FROM OrderDetails} \\ \textbf{JOIN Products ON OrderDetails} \, . \, \textbf{ProductID} \, = \, \textbf{Products} \, . \, \textbf{ProductID} \\ \textbf{GROUP BY OrderDetails} \, . \, \textbf{OrderID} \end{array}
```

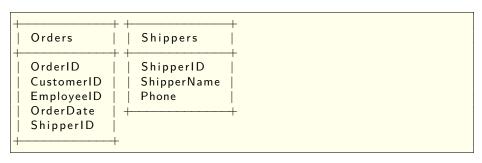
←□ → ←□ → ←□ → ←□ → ←□ → ←□



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
```



Pour chaque Shippers, combien de commandes?

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

Pour chaque Shippers, combien de commandes?

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

33 / 39

Filtrage des agrégats

Quelques trucs en plus

- ▶ On peut jointer une table avec elle même
- On peut jointer sur plusieurs égalités
- On peut faire des requêtes emboitées

Quelques trucs en plus

- ▶ On peut jointer une table avec elle même
- On peut jointer sur plusieurs égalités
- On peut faire des requêtes emboitées

35 / 39

Quelques trucs en plus

- On peut jointer une table avec elle même
- On peut jointer sur plusieurs égalités
- On peut faire des requêtes emboitées

35 / 39

Table Randonnee

```
ld 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 :
```

36 / 39

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.
```

37 / 39

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.

ldentifier 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.

Ecrire 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.

Ecrire 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.

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 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 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 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 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. Id Auteur
```

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 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 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
```

39 / 39