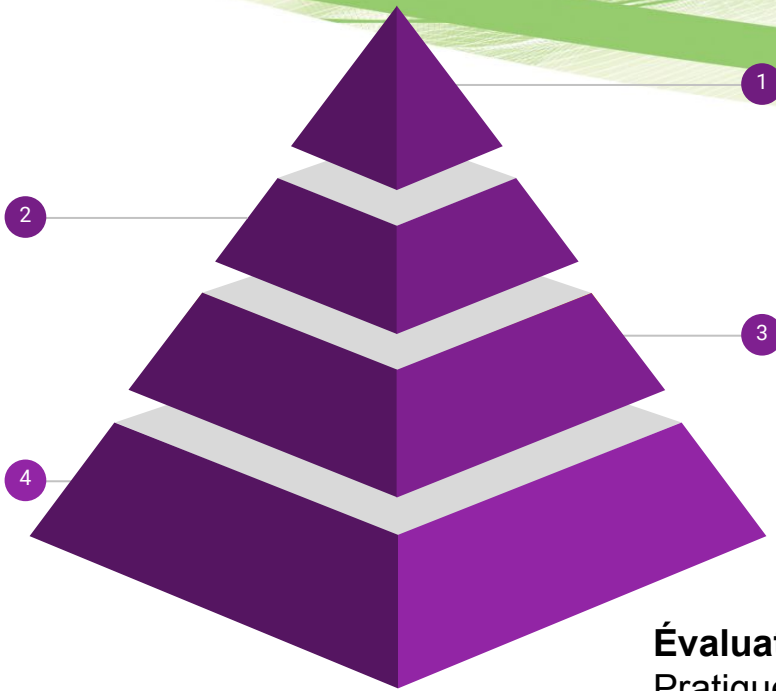
Qualité et au-delà du relationnel

S4 - R4.03 - 2022/2023

Plan de ce cours



1. **Base de données relationnelle :**
   Dépendances fonctionnelles
   Normalisation
2. **Introduction aux bases de données non relationnelles**
3. **MongoDB**
4. **MongoDB + NodeJS**

Évaluation:
Pratique ==> projet
Théorique ==> évaluation écrite / QCM

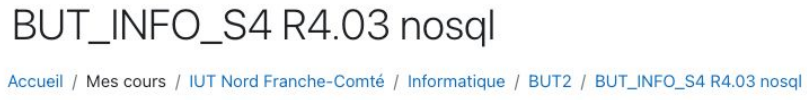
Instructeurs et ressources

Qui contacter :

- Joseph Azar
  - joseph.azar@univ-fcomte.fr
- Christophe Dehe
  - christophe.dehe@univ-fcomte.fr

Où télécharger les ressources ?

- Vérifiez sur cours-info (S4)
- Moodle: IUT Nord Franche-Comté / Informatique / BUT2 / **BUT_INFO_S4 R4.03 nosql**
  - mot de passe pour accéder au cours : **but2nosql23**



Objectif de cette séance

I. Définir l'intégrité des données et son importance
II. Introduction aux dépendances fonctionnelles
    A. Règles de dépendances fonctionnelles
    B. Types de dépendances fonctionnelles dans le SGBD
III. Activité 1 : exercices sur les sujets abordés

Objectif de cette séance

Pourquoi se soucier de l'intégrité des données ?

- L'intégrité des données fait référence à l'exactitude et à la cohérence des données stockées dans une base de données.
- L'intégrité des données garantit que les données sont complètes, exactes et fiables, et qu'elles sont conformes aux règles et contraintes définies par la base de données.
- L'intégrité des données est importante car elle garantit que les données sont fiables et qu'elles peuvent être utilisées pour la prise de décision et l'analyse.

Objectif de cette séance

Conséquences d'une mauvaise intégrité des données

Une mauvaise intégrité des données peut entraîner un certain nombre de problèmes, notamment :

- Résultats incorrects ou non fiables des requêtes et de l'analyse.
- Difficultés à identifier et corriger les erreurs.
- Perte de confiance dans les données.
- Augmentation des coûts de nettoyage et de maintenance des données.

Objectif de cette séance

Le rôle des dépendances fonctionnelles et de la normalisation dans le maintien de l'intégrité des données

- Les dépendances fonctionnelles et la normalisation sont des techniques utilisées **pour maintenir l'intégrité des données** dans une base de données relationnelle.
- Les dépendances fonctionnelles **définissent les relations entre les attributs d'une table**, et la normalisation est le processus d'organisation des données dans des tables séparées **pour minimiser la redondance des données** et améliorer l'intégrité des données.
- Ensemble, ces techniques permettent de **garantir que les données sont exactes, cohérentes et fiables**.

Types d'intégrité des données

- **Intégrité de l'entité** : cela définit chaque ligne comme unique dans la table. Deux lignes ne peuvent pas porter les mêmes données. Pour ce faire, une clé primaire peut être définie. Ce champ contient un identifiant unique — deux lignes ne peuvent pas contenir le même identifiant unique.
- **Intégrité référentielle** : l'intégrité référentielle concerne les relations. Lorsque deux ou plusieurs tables ont une relation, nous devons nous assurer que la valeur de la clé étrangère correspond toujours à la valeur de la clé primaire, sinon l'enregistrement est orphelin. L'intégrité référentielle empêche l'ajout d'enregistrements à une table liée s'il n'y a pas d'enregistrement associé dans la table primaire ou parente, ou la suppression d'enregistrements d'une table parente s'il existe des enregistrements liés correspondants.

Types d'intégrité des données

- **Intégrité du domaine :** cela concerne principalement la validité des entrées pour une colonne donnée. Il est important de sélectionner le type de données approprié pour une colonne et de définir les contraintes appropriées pour définir le format des données ou restreindre la plage de valeurs possibles.
- **Intégrité définie par l'utilisateur :** dans certains cas, vous pouvez appliquer des règles métier à la base de données. Ces règles peuvent ne pas avoir été couvertes par les trois types d'intégrité. L'intégrité définie par l'utilisateur vous aide à appliquer vos propres règles pour garantir l'intégrité des données.

Un Exemple pour l'échauffement

On a les données suivantes :

- Ensembles de clients avec un numéro, un nom et une ville.
- Pour chaque client, un ensemble de commandes avec un numéro de commande, un produit commandé et une date commande.

Exemple de valeurs :

Clients	**Commandes**
145, Joseph, Belfort	<C01, Led Zeppelin, Dec-2022>, <C02, Doors, Jan-2023>
146, Christophe, Bessoncourt	<C03, Santana, Nov-2022>, <C04, Pink Floyd, Dec-2022>, <C05, Led Zeppelin, Oct-2022>

Un Exemple pour l'échauffement

On met dans une table les attributs d'un client et les commandes. On considère une commande comme un attribut.

NumC	NomC	Ville	Com1	Com2	Com3	Com4
145	Joseph	Belfort	C01, Led Zeppelin, Dec-2022	C02, Doors, Jan-2023		
146	Christophe	Bessoncourt	C03, Santana, Nov-2022	C04, Pink Floyd, Dec-2022	C05, Led Zeppelin, Oct-2022	

Un Exemple pour l'échauffement

Q: Donnez deux problèmes pour cette solution.

NumC	NomC	Ville	Com1	Com2	Com3	Com4
145	Joseph	Belfort	C01, Led Zeppelin, Dec-2022	C02, Doors, Jan-2023		
146	Christophe	Bessoncourt	C03, Santana, Nov-2022	C04, Pink Floyd, Dec-2022	C05, Led Zeppelin, Oct-2022	

Un Exemple pour l'échauffement

NumC	NomC	Ville	Com1	Com2	Com3	Com4
145	Joseph	Belfort	C01, Led Zeppelin, Dec-2022	C02, Doors, Jan-2023		
146	Christophe	Bessoncourt	C03, Santana, Nov-2022	C04, Pink Floyd, Dec-2022	C05, Led Zeppelin, Oct-2022	

Exemple de problèmes :

- On ne peut accéder aux noms des CD commandés ou aux dates de commandes.
- On est obligé de supposer un nombre maximum de commande (4 ici)

Un Exemple pour l'échauffement

Autre solution : On décompose les champs de commandes pour en faire des attributs de la relation (on élargit la table).

NumC	NomC	Ville	NumCom1	Produit1	Date1
145	Joseph	Belfort	C01	Led Zeppelin	Dec-2022
146	Christophe	Bessoncourt	C03	Santana	Nov-2022

NumCom2	Produit2	Date2
C02	Doors	Jan-2023
C04	Pink Floyd	Dec-2022

Un Exemple pour l'échauffement

Q: Donnez cinq problèmes pour cette solution.

NumC	NomC	Ville	NumCom1	Produit1	Date1
145	Joseph	Belfort	C01	Led Zeppelin	Dec-2022
146	Christophe	Bessoncourt	C03	Santana	Nov-2022

NumCom2	Produit2	Date2
C02	Doors	Jan-2023
C04	Pink Floyd	Dec-2022

Un Exemple pour l'échauffement

Q: Donnez cinq problèmes pour cette solution.

- La Table peut paraître très large.

- On est encore obligé de supposer un nombre maximum de commandes (4 ici).

- Beaucoup de valeurs indéfinies (Il n'y a pas toujours 4 commandes).

- Surtout, requête « Nom du Client qui a commandé Doors ?»

SELECT NomC FROM table WHERE Produit1= "Doors" OR Produit2= "Doors" OR Produit3= "Doors" OR Produit4= "Doors" **Incommode !**

- On ne peut pas, par exemple, calculer la somme ou la moyenne des prix des commandes du client '145'

SELECT (Prix1 + Prix2 + Prix3) / 3 FROM table WHERE NumC="145" **Incommode !**

Un Exemple pour l'échauffement

Autre solution : On met les commandes d'un même client dans autant de lignes qu'il faut.

NumCom	Produit	Date	NumC	NomC	Ville
C01	Led Zeppelin	Dec-2022	145	Joseph	Belfort
C02	Doors	Jan-2023	145	Joseph	Belfort
C03	Santana	Nov-2022	146	Christophe	Bessoncourt
C04	Pink Floyd	Dec-2022	146	Christophe	Bessoncourt
C05	Led Zeppelin	Oct-2022	146	Christophe	Bessoncourt

Un Exemple pour l'échauffement

On constate que les problèmes précédents ont disparu :

- Pas de limites pour les commandes d'un client.

- Pas de valeurs indéfinies.

- « Nom du Client qui a commandé Doors ?» La valeur 'Doors' apparaît dans LA colonne Produit

SELECT NomC FROM table WHERE Produit= "Doors"

- On peut calculer le nombre de commandes pour un client donné (on ne pouvait pas avant)

SELECT count (*) FROM table WHERE NumC="145"

- Ou la moyenne des prix. Plus simple cette fois-ci : colonne unique Prix

SELECT Avg (Prix) FROM table WHERE NumC="145"

Un Exemple pour l'échauffement

Q: Néanmoins, cette table souffre encore de quelques anomalies. Citez 3 problèmes.

NumCom	Produit	Date	NumC	NomC	Ville
C01	Led Zeppelin	Dec-2022	145	Joseph	Belfort
C02	Doors	Jan-2023	145	Joseph	Belfort
C03	Santana	Nov-2022	146	Christophe	Bessoncourt
C04	Pink Floyd	Dec-2022	146	Christophe	Bessoncourt
C05	Led Zeppelin	Oct-2022	146	Christophe	Bessoncourt

Un Exemple pour l'échauffement

Q: Néanmoins, cette table souffre encore de quelques anomalies. Citez 3 problèmes.

Redondances : mêmes informations client sur plusieurs lignes.

- Perte d'espace.

- Risque d'inconsistance. La requête suivante modifie la ville du client qui a fait la commande 'C01'

```
UPDATE table
SET Ville = "Montbéliard"
WHERE NumCom = "C01"
```

La base est dans un état inconsistant. Joseph habite dans quelle ville Belfort ou Montbéliard ?

Un Exemple pour l'échauffement

Q: Néanmoins, cette table souffre encore de quelques anomalies.

NumCom	Produit	Date	NumC	NomC	Ville
C01	Led Zeppelin	Dec-2022	145	Joseph	Belfort
C02	Doors	Jan-2023	145	Joseph	Belfort
C03	Santana	Nov-2022	146	Christophe	Bessoncourt
C04	Pink Floyd	Dec-2022	146	Christophe	Bessoncourt
C05	Led Zeppelin	Oct-2022	146	Christophe	Bessoncourt
C06	Bohemian Rhapsody	Jan-2023	148	David	Belfort

Un Exemple pour l'échauffement

Q: Néanmoins, cette table souffre encore de quelques anomalies.

- Plus encore, supposer qu'on supprime la seule commande faite par un client (par exemple David) :

DELETE FROM table WHERE NumCom="C06"

On a perdu aussi les informations sur le client "David".

- Inversement, on ne peut pas rajouter un nouveau client, qui n'a encore aucune commande. On ne peut avoir NumCom indéfini, car c'est la clé primaire de la table.

Un Exemple pour l'échauffement

Autre solution : deux tables, une pour les clients **Client** et une pour les commandes **Commande.**

NumCom	Produit	Date	NumC
C01	Led Zeppelin	Dec-2022	145
C02	Doors	Jan-2023	145
C03	Santana	Nov-2022	146
C04	Pink Floyd	Dec-2022	146
C05	Led Zeppelin	Oct-2022	146

Commande

NumC	NomC	Ville
145	Joseph	Belfort
146	Christophe	Besoncourt

Client

On constate que les problèmes précédents ont disparu.

Un Exemple pour l'échauffement

Jusqu'à présent, vous devriez être familier avec tout cela.

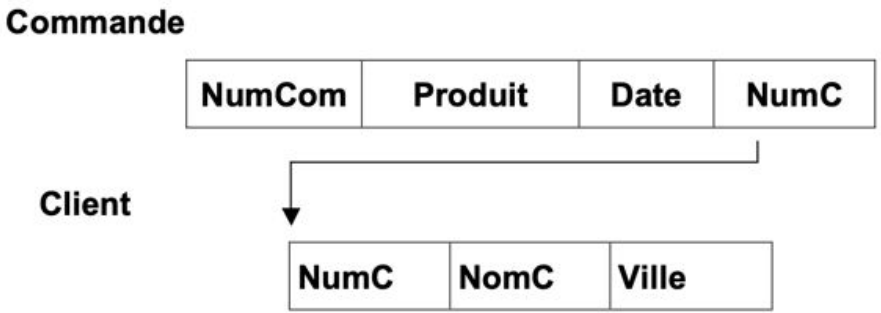
Règles simples:

1. Chaque type d'entité doit être représenté par une table.
2. La clé primaire d'une table est l'identifiant du type d'entité.
3. Les champs d'une table sont constitués des autres propriétés du type d'entité
    a. Mais les types d'entités ne sont pas indépendants. Il y a une « relation » c'est-à-dire une association (ang. relationship) entre les clients et les commandes : un client fait des commandes. Dans cet exemple, la relation est de type 1-M (1-à-plusieurs) entre clients et commandes. Un client fait plusieurs (0, 1 ou n) commandes, et une commande est faite par un seul client. Autrement dit, la relation est une fonction entre l'ensemble des commandes et l'ensemble des clients.

Un Exemple pour l'échauffement

Pour représenter cette association dans la base de données, l'identifiant de client (partie-1 de l'association) est à rajouter comme champ dans la table commande (partie-M de l'association).

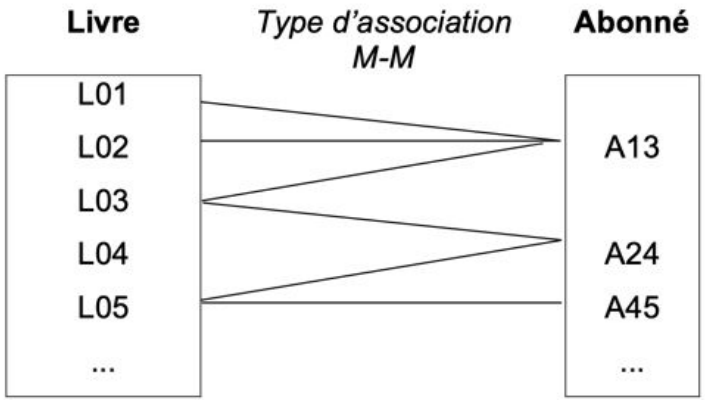
Pour chaque association, l'identifiant du type d'entité partie-1 est à rajouter comme champ dans la table correspondant au type d'entité partie-M.



Un Exemple pour l'échauffement

Autre exemple : cas de relation M-M

Considérons maintenant la base de données sur les livres de prêt les abonnés. Ici, le type d'association entre un abonné et un livre est de type plusieurs à plusieurs (M-M). Un livre peut être emprunté par plusieurs abonnés, et un abonné peut emprunter plusieurs livres



Un Exemple pour l'échauffement

Dans ce cas, les règle 1 à 3 sont les mêmes, on a une table par type d'entité, donc une table **Livre** et une table **Abonné**.

Livre

NumInv	Titre	Auteur	Qte

Abonne

NumAb	Nom	Prénom

Pour la relation de prêt entre un abonné et un livre, on crée une autre table avec comme champs :

- les identifiants des types d'entités associés,
- tout autre champ qui dépend des deux entités, comme la date de prêt ici.

Un Exemple pour l'échauffement

Livre

NumInv	Titre	Auteur	Qte

Abonne

NumAb	Nom	Prénom

Pret

NumInv	NumAb	DatePret

D'où la règle 4:

Pour chaque association M-M, ajouter une table avec les identifiants des types d'entités associés et tout autre attribut qui dépend de ces deux types d'entités pris ensemble. La clé primaire de cette table est la combinaison des deux identifiants.

Formalisation : Notion de dépendance fonctionnelle

Soit R une relation avec les attributs A, B et C.

R (A, B, C)

Définition : On dit qu'il y a une Dépendance Fonctionnelle entre les attributs A et B si la projection de R [A, B] est une fonction de A vers B, notée

A ⇒ B

On dit aussi que A détermine B, i.e. si on connaît la valeur de A dans R, on peut connaître aussi la valeur de B.

Formalisation : Notion de dépendance fonctionnelle

Une dépendance fonctionnelle (FD) est une **relation entre deux attributs**, généralement entre le PK et d'autres attributs non clés dans une table. Pour toute relation R, l'attribut Y est fonctionnellement dépendant de l'attribut X (généralement le PK), si pour chaque instance valide de X, cette valeur de X détermine de manière unique la valeur de Y. Cette relation est indiquée par la représentation ci-dessous :

X ———> Y

Le côté gauche du diagramme FD ci-dessus est appelé le **déterminant** et le côté droit en est le **dépendant**.

La dépendance fonctionnelle aide à maintenir la qualité des données dans la base de données. Elle joue un rôle essentiel pour trouver la différence entre une bonne et une mauvaise conception de base de données.

Dépendance Fonctionnelle : Exemples

R

A	*B*	*C*
a	*e*	*b*
a	*e*	*c*
c	*b*	*d*
g	*e*	*f*

Dépendance Fonctionnelle : Exemples

Ici, **A** détermine **B**, **A ⇒ B**, mais ne détermine pas **C** (**A -/-> C**). On peut aussi constater que **C ⇒ B** et **C ⇒ A**.

Proposition : Dans une relation, la clé primaire détermine tous les autres attributs.

Cela découle du fait que la valeur de la clé identifie un tuple. Donc, les autres composants du tuple. Dans la relation

Client (**NumC**, **NomC**, **Ville**), on a :

NumC ⇒ NomC

et

NumC ⇒ Ville

R

A	*B*	*C*
a	*e*	*b*
a	*e*	*c*
c	*b*	*d*
g	*e*	*f*

Dépendance Fonctionnelle : Exemples

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeeSalary)

Une dépendance fonctionnelle existe entre **EmployeeID** et **EmployeeName**, ce qui signifie que pour un **EmployeeID** donné, il n'existe qu'un seul **EmployeeName** correspondant. Cela signifie que si nous connaissons l'identité d'un employé, nous connaissons son nom. Cela peut être écrit comme suit :

$$\textbf{EmployeeID} \Rightarrow \textbf{EmployeeName}$$

Dépendance Fonctionnelle : Exemples

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeeSalary)

Un autre exemple de dépendance fonctionnelle dans la même table pourrait être :

EmployeeID ⇒ EmployeeAddress et EmployeeSalary

Cela signifie que pour un **EmployeeID** donné, il n'y a qu'un seul **EmployeeAddress** et **EmployeeSalary** correspondants. Cela signifie que si nous connaissons l'identité d'un employé, nous connaissons également son adresse et son salaire.

Il est important de noter que les dépendances fonctionnelles sont utilisées pour définir les relations entre les attributs d'une table et qu'elles contribuent à garantir l'intégrité des données en empêchant la redondance et les incohérences des données.

Dépendance Fonctionnelle : Exemples

Considérez cette relation :

Sales (ProductID, ProductName, ProductPrice, CustomerName)

Cette table contient de mauvaises dépendances fonctionnelles et ne respecte pas l'intégrité des données. ***Pourquoi ?***

Dépendance Fonctionnelle : Exemples

Considérez cette relation :

Sales (ProductID, ProductName, ProductPrice, CustomerName)

Une mauvaise dépendance fonctionnelle dans cette table serait :

ProductName ⇒ ProductPrice

Cela signifie que pour un **ProductName** donné, il n'y a qu'un seul **ProductPrice** correspondant. Cependant, ce n'est pas vrai dans la réalité, car différents produits portant le même nom peuvent avoir des prix différents. Cette dépendance fonctionnelle crée des incohérences et des inexactitudes dans les données, et elle ne respecte pas l'intégrité des données.

Dépendance Fonctionnelle : Exemples

Considérez cette relation :

Sales (ProductID, ProductName, ProductPrice, CustomerName)

Un autre exemple de mauvaise dépendance fonctionnelle dans la même table pourrait être :

CustomerName ⇒ ProductID, ProductName

Cela signifie que pour un nom de client donné, il n'y a qu'un seul identifiant de produit et un seul nom de produit correspondants. Mais en réalité, un client peut avoir plusieurs produits. Cette dépendance fonctionnelle crée des incohérences et des inexactitudes dans les données, et elle ne respecte pas l'intégrité des données.

Dépendance Fonctionnelle : Exemples

Name	Class	Subject	Age
Pooja	5th	English	10
Priya	4th	Hindi	9
Pooja	5th	Maths	10
Pooja	6th	Science	10
Sneha	7th	Computer	11

2. Class->Name, Class+Subject->Name+Subject
3. Class->Name, Name-> Age, Class->Age
4. Class->Name, Class->Age, Class->Name+Age
5. Class->Name+Age, Class-> Name, Class->Age
6. Class->Name, Name+Subject->Age, Class+subject->Age

Dépendance Fonctionnelle : Mots clés

Voici quelques termes clés pour la dépendance fonctionnelle dans la base de données :

Mot-clé	**Description**
Axiome	Les axiomes sont un ensemble de règles d'inférence utilisées pour déduire toutes les dépendances fonctionnelles sur une base de données relationnelle.
Décomposition	C'est une règle qui suggère que si vous avez une table qui semble contenir deux entités déterminées par la même clé primaire, vous devriez envisager de les diviser en deux tables différentes.
Dépendant	Il est affiché sur le côté droit du diagramme de dépendance fonctionnelle.
Déterminant	Il est affiché sur le côté gauche du diagramme de dépendance fonctionnelle.
Union	Cela suggère que si deux tables sont séparées et que le PK est le même, vous devriez envisager de les mettre. ensemble.

Dépendance Fonctionnelle : Règles d'inférence

Les axiomes d'Armstrong sont un ensemble de règles d'inférence utilisées pour déduire toutes les dépendances fonctionnelles sur une base de données relationnelle. Ils ont été développés par William W. Armstrong.

Vous trouverez ci-dessous les règles d'inférence les plus importantes pour la dépendance fonctionnelle dans la base de données :

- Réflexivité
- Accroissement / augmentation
- Transitivité
- Pseudo-transitivité
- Union
- Décomposition

Dépendance Fonctionnelle : Réflexivité

Si Y est un sous-ensemble de X, alors X détermine fonctionnellement Y. Cela signifie que si Y dépend de X, alors X détermine Y.

$$\text{If } Y \subseteq X, \text{ then } X \rightarrow Y$$

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone)

Nous pouvons en déduire que la dépendance fonctionnelle **EmployeeID ⇒ EmployeeName** est vraie, car pour un **EmployeeID** donné, il n'y a qu'un seul **EmployeeName** correspondant. Cela signifie que nous pouvons déterminer le nom d'un employé en fonction de son ID.

Dépendance Fonctionnelle : Réflexivité

$$\text{If } Y \subseteq X, \text{ then } X \rightarrow Y$$

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone)

Nous pouvons également en déduire que la dépendance fonctionnelle

EmployeeID ⇒ EmployeeID est vraie, car pour un **EmployeeID** donné, il n'y a qu'un seul **EmployeeID** correspondant. Cela signifie que chaque employé a un identifiant unique.

Maintenant, selon la propriété de réflexivité des dépendances fonctionnelles, puisque **EmployeeID** est un sous-ensemble de **{EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone}**, alors **{EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone}** détermine fonctionnellement **EmployeeID**. En d'autres termes, nous pouvons déterminer **l'EmployeeID** en fonction de n'importe quel sous-ensemble des attributs de la table qui inclut **EmployeeID**.

Dépendance Fonctionnelle : Augmentation

L'axiome d'augmentation, également connu sous le nom de dépendance partielle, dit que si X détermine Y, alors XZ détermine YZ pour tout Z.

$$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$$

Cela signifie que si A détermine B et que nous ajoutons un ensemble d'attributs C, le nouvel ensemble d'attributs AC déterminera toujours l'ensemble d'attributs BC. Cette propriété nous permet d'inférer de nouvelles dépendances fonctionnelles en ajoutant des attributs aux dépendances existantes.

Dépendance Fonctionnelle : Augmentation

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone)

Nous pouvons en déduire que la dépendance fonctionnelle **EmployeeID ⇒ EmployeeName** est vraie, car pour un **EmployeeID** donné, il n'y a qu'un seul **EmployeeName** correspondant. Cela signifie que nous pouvons déterminer le nom d'un employé en fonction de son ID.

Maintenant, selon l'axiome d'augmentation, si nous avons la dépendance fonctionnelle **EmployeeID -> EmployeeName**, et C est un ensemble d'attributs, nous pouvons également en déduire que

EmployeeID, EmployeeAddress ⇒ EmployeeName, EmployeePhone

est vrai. Cela signifie que si nous connaissons **EmployeeID** et **EmployeeAddress**, nous pouvons également déterminer **EmployeeName** et **EmployeePhone**.

Dépendance Fonctionnelle : Dépendance Partielle

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone)

Considérez la dépendance fonctionnelle

EmployeeID ⇒ EmployeeAddress, EmployeePhone

Nous pouvons voir que **EmployeeAddress** et **EmployeePhone** dépendent de **EmployeeID**, mais **EmployeeAddress** et **EmployeePhone** ne dépendent pas l'un de l'autre. Cela signifie que **EmployeeAddress** et **EmployeePhone** ne dépendent pas entièrement de **EmployeeID**, c'est ce qu'on appelle une dépendance partielle.

Une dépendance partielle se produit lorsqu'un attribut ou un ensemble d'attributs dépend de la clé primaire mais ne dépend pas des autres attributs de clé non primaire. Dans ce cas, **EmployeeAddress** et **EmployeePhone** dépendent tous deux de **EmployeeID**, mais ils ne dépendent pas l'un de l'autre.

Dépendance Fonctionnelle : Dépendance Partielle

Cette dépendance partielle peut entraîner **une redondance des données** et des **anomalies de mise à jour**, car la modification de la valeur d'un attribut non clé peut ne pas affecter la valeur de l'autre attribut non clé. Pour résoudre ce problème, il est préférable de diviser la table et de placer **EmployeeAddress** et **EmployeePhone** dans des tables séparées et de les lier à la table **Employee** en utilisant **EmployeeID** comme clé étrangère.

En résumé, une dépendance partielle se produit lorsqu'un attribut ou un ensemble d'attributs dépend de la clé primaire mais ne dépend pas des autres attributs de clé non primaire, ce qui entraîne une redondance des données et des anomalies de mise à jour. Elle peut être résolue en utilisant la **normalisation**.

Dépendance Fonctionnelle : Transitivité

Si un attribut X détermine un attribut Y et que cet attribut Y détermine un autre attribut Z, alors X détermine Z. Soient X, Y et Z des attributs :

$$X \rightarrow Y \text{ et } Y \rightarrow Z \Rightarrow X \rightarrow Z$$

Considérez cette relation :

Employee (EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone)

Considérez les dépendances fonctionnelles :

EmployeeID ⇒ EmployeeAddress

EmployeeAddress ⇒ EmployeePhone

Dépendance Fonctionnelle : Transitivité

La transitivité signifie que si **X ⇒ Y** et **Y ⇒ Z** sont vrais, alors **X ⇒ Z** est également vrai. Dans ce cas, **EmployeeID** détermine **EmployeeAddress** et **EmployeeAddress** détermine **EmployeePhone**, donc **EmployeeID** détermine **EmployeePhone**.

La transitivité peut créer une redondance des données et des anomalies de mise à jour (**UPDATE ANOMALIES**), car elle crée une dépendance fonctionnelle qui pourrait ne pas exister dans le monde réel. Pour résoudre ce problème, il est préférable de diviser la table et de placer **EmployeeAddress** et **EmployeePhone** dans des tables séparées et de les lier à la table **Employee** en utilisant **EmployeeID** comme clé étrangère.

En résumé, la transitivité de la dépendance fonctionnelle entraîne une redondance des données et des anomalies de mise à jour. **Elle peut être résolue en utilisant la normalisation.**

Dépendance Fonctionnelle : Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs. Soient X, Y et Z des attributs :

$$X \rightarrow Y \text{ et } X \rightarrow Z \Rightarrow X \rightarrow YZ$$

Considérez deux tables distinctes, "**Employees**" et "**EmployeeContacts**" avec les attributs suivants :

Table "**Employees**": <**EmployeeID, EmployeeName, EmployeeAddress**>

Table "**EmployeeContacts**": <**EmployeeID, EmployeePhone, EmployeeEmail**>

Dépendance Fonctionnelle : Union

Selon la définition de la dépendance d'union, si **X** détermine **Y** et **X** détermine **Z**, alors **X** doit également déterminer **Y** et **Z**. Dans ce cas, **EmployeeID** détermine **EmployeeName** dans la table **Employees** et **EmployeePhone** et **EmployeeEmail** dans la table **EmployeeContacts**.

Donc, sur la base de cette définition, nous devrions envisager de regrouper ces deux tables, car **EmployeeID** est la clé primaire des deux tables et détermine à la fois **EmployeeName**, **EmployeePhone** et **EmployeeEmail**.

En combinant ces tables, vous pouvez **éviter la redondance des données et les anomalies de mise à jour**, ainsi qu'améliorer les performances des requêtes en réduisant les opérations de jointure. Cela peut être fait en créant une nouvelle table appelée "**Employees**" avec les attributs : **EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone, EmployeeEmail** et utilisez **EmployeeID** comme clé primaire.

Dépendance Fonctionnelle : Décomposition

Si un attribut détermine un groupe d'attribut, alors il détermine chacun des attributs de ce groupe pris individuellement. Soient X, Y et Z des attributs :

$$X \rightarrow YZ \Rightarrow X \rightarrow Z \text{ et } X \rightarrow Y$$

Considérez une table appelée "Employees" avec les attributs suivants : **EmployeeID, EmployeeName, EmployeeAddress, EmployeePhone et EmployeeEmail.**

Nous pouvons en déduire que la dépendance fonctionnelle **EmployeeID ⇒ EmployeeName, EmployeeAddress, EmployeePhone et EmployeeEmail** est vraie, car pour un **EmployeeID** donné, il n'y a qu'un seul **EmployeeName**, **EmployeeAddress**, **EmployeePhone** et **EmployeeEmail** correspondants. Cela signifie que nous pouvons déterminer les informations d'un employé en fonction de son ID.

Dépendance Fonctionnelle : Décomposition

Selon la définition de la dépendance de décomposition, si **X** détermine **Y** et **Z**, alors **X** détermine **Y** et **X** détermine **Z** séparément. Dans ce cas, **EmployeeID** détermine **EmployeeName**, **EmployeeAddress**, **EmployeePhone** et **EmployeeEmail**.

Ainsi, sur la base de cette définition, nous devrions envisager de diviser la table en deux tables ou plus, car **EmployeeID** détermine plusieurs entités (**EmployeeName**, **EmployeeAddress**, **EmployeePhone** et **EmployeeEmail**) séparément.

En décomposant la table, vous pouvez améliorer l'intégrité des données et éviter la redondance des données. Cela peut être fait en créant deux nouvelles tables appelées "**EmployeeInformation**" avec les attributs : **EmployeeID, EmployeeName, EmployeeAddress** et "**EmployeeContacts**" avec les attributs : **EmployeeID, EmployeePhone, EmployeeEmail**, et utilisez **EmployeeID** comme clé primaire dans les deux tables.

Dépendance Fonctionnelle : Union et Décomposition

Les dépendances fonctionnelles d'union et de décomposition peuvent sembler contradictoires à première vue. La décision d'utiliser des dépendances fonctionnelles d'union ou de décomposition **dépend des exigences et des contraintes spécifiques de votre modèle de données.**

La dépendance fonctionnelle d'union est généralement utilisée lorsque **les données des tables séparées sont étroitement liées**, et il est logique de les stocker ensemble dans une seule table pour faciliter les requêtes et l'intégrité des données. La dépendance fonctionnelle de l'union peut également être utilisée dans les cas où **les données sont incomplètes** ou lorsque les données sont stockées dans différentes tables et que toutes les informations ne sont pas disponibles dans une seule table.

La dépendance fonctionnelle de décomposition est généralement utilisée **lorsque les données de la table ne sont pas étroitement liées**, et il est logique de les stocker dans des tables séparées pour une meilleure intégrité des données et une interrogation plus facile. La dépendance fonctionnelle de décomposition peut également être utilisée **lorsque les données sont redondantes**, et cela conduit à des anomalies de mise à jour.

Types de dépendances fonctionnelles dans le SGBD

Il existe principalement quatre types de dépendance fonctionnelle dans le SGBD.
Voici les types de dépendances fonctionnelles dans le SGBD :

- Dépendance multivaluée
- Dépendance fonctionnelle triviale
- Dépendance fonctionnelle non triviale
- Dépendance transitive

Dépendance multivaluée dans le SGBD

La dépendance à valeurs multiples se produit dans la situation où il existe **plusieurs attributs à valeurs multiples indépendants dans une seule table**.

La dépendance à plusieurs valeurs se produit lorsque deux attributs d'une table **sont indépendants l'un de l'autre** mais qu'**ils dépendent tous deux d'un troisième attribut**.

Une dépendance à plusieurs valeurs se compose **d'au moins deux attributs qui dépendent d'un troisième attribut**, c'est pourquoi elle nécessite toujours au moins trois attributs.

Considérez l'exemple suivant pour comprendre.

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

Dépendance multivaluée dans le SGBD

Dans cet exemple, **maf_year** et **color** sont indépendants l'un de l'autre mais dépendent de **car_model**. Dans cet exemple, ces deux colonnes sont dites multivaluée dépendantes de **car_model**.

Cette dépendance peut être représentée comme ceci :

car_model ⇒ maf_year + color

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

Dépendance fonctionnelle triviale dans le SGBD

X ⇒ Y est une dépendance fonctionnelle triviale si Y est un sous-ensemble de X.
Comprenons avec un exemple de dépendance fonctionnelle triviale.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Dépendance fonctionnelle triviale dans le SGBD

Considérez cette table avec deux colonnes **Emp_id** et **Emp_name**.

{Emp_id, Emp_name} ⇒ Emp_id est une dépendance fonctionnelle triviale car **Emp_id** est un sous-ensemble de **{Emp_id,Emp_name}**.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Dépendance fonctionnelle non triviale dans le SGBD

La dépendance fonctionnelle connue sous le nom de dépendance non triviale se produit lorsque **A ⇒ B** est vrai, où **B** n'est pas un sous-ensemble de **A**.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Dépendance fonctionnelle non triviale dans le SGBD

{Company} ⇒ {CEO} (si nous connaissons l'entreprise, nous connaissons le nom du CEO)

Mais **{CEO}** n'est pas un sous-ensemble de **{Company}**, et il s'agit donc d'une dépendance fonctionnelle non triviale.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Dépendance transitive dans le SGBD

Une dépendance transitive est un type de dépendance fonctionnelle qui se produit lorsque "**T**" est indirectement formé par deux dépendances fonctionnelles. Comprenons avec l'exemple de dépendance transitive suivant:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Dépendance transitive dans le SGBD

{Company} ⇒ {CEO} (si nous connaissons l'entreprise, nous connaissons le nom du CEO)

{CEO } ⇒ {Age} Si nous connaissons le CEO, nous connaissons l'âge

Donc selon la règle de dépendance transitive :

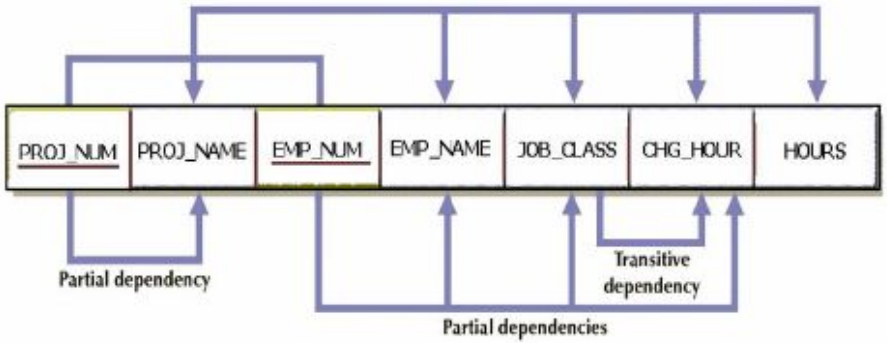
{Company} ⇒ {Age} devrait tenir, cela a du sens car si nous connaissons le nom de l'entreprise, nous pouvons connaître l'âge du CEO.

Remarque : Vous devez vous rappeler que la dépendance transitive ne peut se produire que dans une relation de trois attributs ou plus.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Diagramme de dépendance

Un diagramme de dépendances illustre les différentes dépendances qui peuvent exister dans une table contenant une redondance de données.



Q: Identifiez les dépendances dans cette table.

Diagramme de dépendance

PROJ_NUM et **EMP_NUM**, combinés, sont les **PK**.

Dépendances partielles :

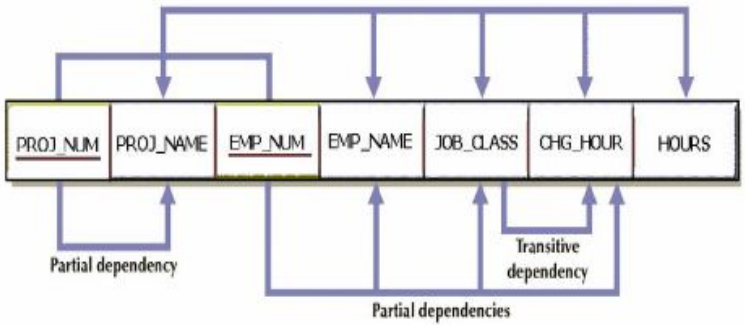
PROJ_NUM ⇒ PROJ_NAME

EMP_NUM ⇒ EMP_NAME, **JOB_CLASS**

EMP_NUM ⇒ CHG_HOUR

Dépendance transitive :

JOB_CLASS ⇒ CHG_HOUR



Conclusion

- La dépendance fonctionnelle se produit lorsqu'un attribut détermine un autre attribut dans un système SGBD.
- Comprendre la dépendance fonctionnelle évite la redondance des données. Par conséquent, les mêmes données ne se répètent pas à plusieurs endroits dans la base de données.
- Elle vous aide à maintenir la qualité des données dans la base de données.
- Axiome, Décomposition, Dépendant, Déterminant, Union sont des termes clés pour la dépendance fonctionnelle.
- Les anomalies causées par de mauvaises dépendances fonctionnelles pourraient être résolues en utilisant la normalisation.

Exercice 1

Soit la relation R suivante :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Question 1 : Proposez une clé primaire pour cette relation. Justifiez brièvement.

Exercice 1 : solution

Soit la relation R suivante :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Réponse 1 : Il y a trois clés candidates : {B,C}, {B,E} et {B,G}, soit la concaténation des colonnes B et C, ou B et E ou B et G. Ce sont en effet les plus petites combinaisons qui sont uniques pour cette relation, et donc qui permettent de distinguer deux relevés. Pour toutes les autres combinaisons, soit elles ne sont pas uniques, soit elles contiennent {B,C}, {B,E} ou {B,G}.

La clé primaire peut donc être choisie parmi ces trois candidats.

Exercice 1

Soit la relation R suivante :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Question 2 : Cette relation contient-elle des redondances ? Si oui lesquelles ? Justifiez brièvement.

Exercice 1 : solution

Soit la relation R suivante :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Réponse 2 : La relation contient des redondances : les colonnes A, D et F d'une part et E et G d'autre part sont redondantes. En effet pour une valeur donnée de A, on obtient toujours les mêmes valeurs de D et F et pour une valeur donnée de E on obtient toujours la même valeur de G.

Exercice 1

Soit la relation R suivante :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Question 3 : Si la relation contient des redondances, proposez une solution contenant exactement la même information, mais sans redondance.

Exercice 1 : solution

Réponse 3 : La seule solution pour supprimer les redondances est de découper la relation R en relations non redondantes.

R1

A	B	C	E
0	1	1	5
0	2	1	9
0	1	2	6
0	1	3	7
1	2	3	7
0	3	3	9
1	4	3	8
1	1	4	9

R2

A	D	F
0	10	X
1	20	Y

R3

E	G
5	A
9	G
6	S
7	D
8	F

Exercice 2

Soit la table suivante :

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

Question : Donnez deux anomalies d'insertion, une anomalie de suppression et une anomalie de mise à jour.

Exercice 2 : solution

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

- **Insérer un nouvel employé dans EmpDept**
  - ✷ compatibilité des données du département à vérifier avec les autres tuples de EmpDept
  - ✷ insérer des valeurs nulles si aucune donnée de département n'est entrée
- **Insérer un nouveau département sans employé dans EmpDept**
  - ✷ tuple avec des valeurs nulles pour les données des employés (mais le SSN est la clé !)
  - ✷ supprimer les nuls (insertion complexe) lorsque le premier employé du département est inséré

Exercice 2 : solution

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

Supprimer d'EmpDept le dernier employé d'un département

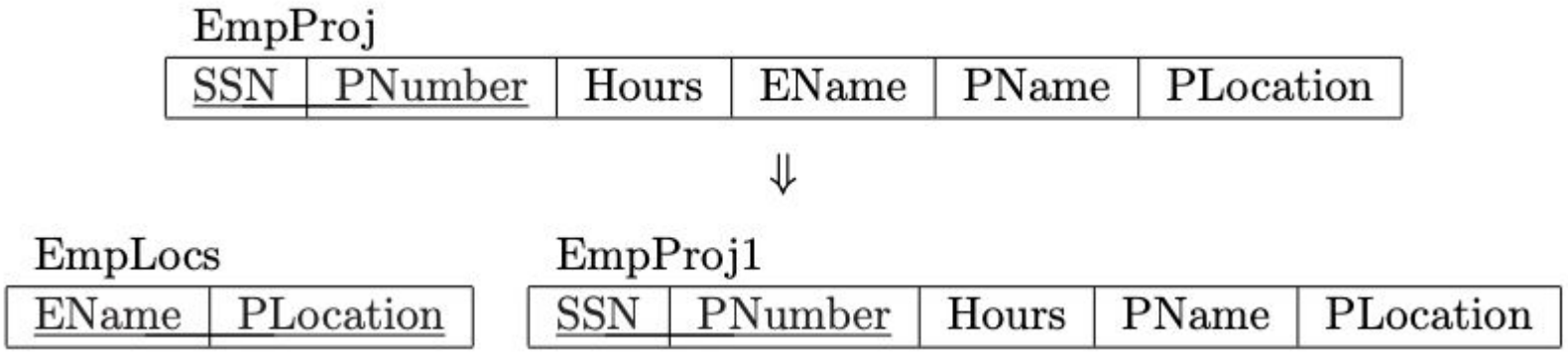
→ perte d'informations ou mise à jour complexe

Changer de responsable d'un département

→ affecte plusieurs tuples pour éviter les incohérences

Exercice 3

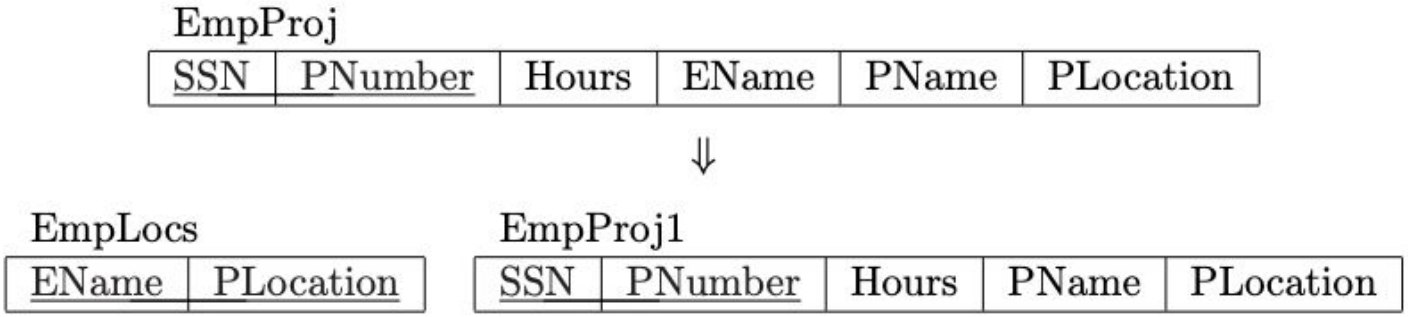
Considérons la décomposition suivante :



Propriété de jointure sans perte : garantit des résultats significatifs pour les jointures.

Question : Cette décomposition donne-t-elle des résultats significatifs pour les jointures ? Expliquer.

Exercice 3 : solution



EmpProj /= jointure de ses projections **EmpLocs** et **EmpProj1**:

- l'attribut commun (**PLocation**) n'est pas une clé ou une clé étrangère
- la jointure donne plus de tuples que dans **EmpProj**

Exercice 3-b

Considérons la décomposition suivante :

EmpProj

SSN	P#	Hours	EName	PName	PLocation
1234	1	32.5	Smith	ProdX	Bellaire
1234	2	7.5	Smith	ProdY	Sugarland
6668	3	40.0	Narayan	ProdZ	Houston
4534	1	20.0	English	ProdX	Bellaire
4534	2	20.0	English	ProdY	Sugarland

⇓

EmpProj1

SSN	P#	Hours	PName	PLocation
1234	1	32.5	ProdX	Bellaire
1234	2	7.5	ProdY	Sugarland
6668	3	40.0	ProdZ	Houston
4534	1	20.0	ProdX	Bellaire
4534	2	20.0	ProdY	Sugarland

EmpLocs

EName	PLocation
Smith	Bellaire
Smith	Sugarland
Narayan	Houston
English	Bellaire
English	Sugarland

Question : Donnez la table de jointure EmpProj1 ⋈ EmpLocs et marquez les tuples où il y a une anomalie.

Exercice 3-b : solution

la jointure donne plus de tuples que dans **EmpProj**

EmpProj

SSN	P#	Hours	EName	PName	PLocation
1234	1	32.5	Smith	ProdX	Bellaire
1234	2	7.5	Smith	ProdY	Sugarland
6668	3	40.0	Narayan	ProdZ	Houston
4534	1	20.0	English	ProdX	Bellaire
4534	2	20.0	English	ProdY	Sugarland

⇓

EmpProj1

SSN	P#	Hours	PName	PLocation
1234	1	32.5	ProdX	Bellaire
1234	2	7.5	ProdY	Sugarland
6668	3	40.0	ProdZ	Houston
4534	1	20.0	ProdX	Bellaire
4534	2	20.0	ProdY	Sugarland

EmpLocs

EName	PLocation
Smith	Bellaire
Smith	Sugarland
Narayan	Houston
English	Bellaire
English	Sugarland

EmpProj1 ⋈ EmpLocs

	SSN	P#	Hours	PName	PLocation	EName
	1234	1	32.5	ProdX	Bellaire	Smith
*	1234	1	32.5	ProdX	Bellaire	English
	1234	2	7.5	ProdY	Sugarland	Smith
*	1234	2	7.5	ProdY	Sugarland	English
	6668	3	40.0	ProdZ	Houston	Narayan
*	4534	1	20.0	ProdX	Bellaire	Smith
	4534	1	20.0	ProdX	Bellaire	English
	4534	2	20.0	ProdY	Sugarland	Smith
*	4534	2	20.0	ProdY	Sugarland	English