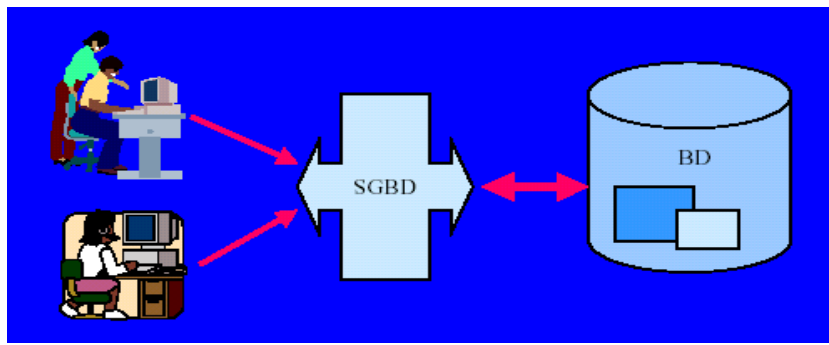


# Développement d'applications de base de données

Myoung-Ah KANG  
kang@isima.fr



- SGBD : Le logiciel qui permet d'interagir avec une BD. Un SGBD est un intermédiaire entre les utilisateurs et les fichiers physiques .

– Un SGBD facilite :

– la gestion de données, avec une représentation intuitive simple sous forme de tables.

– la manipulation de données. On peut insérer, modifier les données et les structures sans modifier les programmes qui manipulent la base de données.

– La plupart des *applications des bases de données* doivent accéder à une base de données pour :

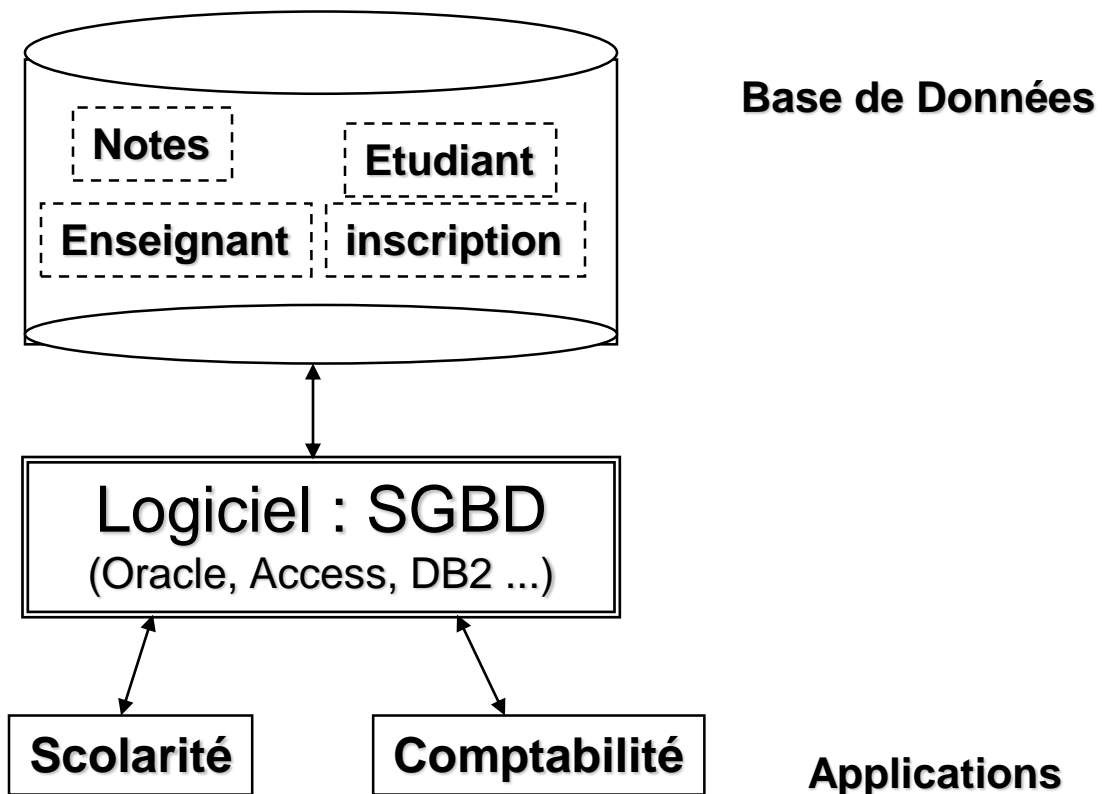
- Récupérer les données et les afficher
- Saisir de nouvelles données
- Mettre à jour les données

# Objectifs d'un SGBD

- Persistance
- Description des données, Manipulation des données (Insérer, Supprimer, Modifier, Interroger)
- Contrôle
  - Partage de données :
    - accès à la même information par plusieurs utilisateurs en même temps
    - gestion des accès concurrents aux données
  - Intégrité des données :
    - définition de contraintes sur les données. Le SGBD veille à ce que toutes les contraintes soient vérifiées à chaque manipulation des données.
    - Ex) l'âge d'une personne  $>0$  ...
  - Confidentialité des données
    - accès aux données autorisés que pour les personnes habilitées
  - Sécurité
    - sécurité physique (sauvegarde), reprise en cas de panne.
- Efficacité de l'accès aux données
  - possibilité de réponses rapides

# Applications de bases de données

- *Application de base de données* : un programme qui utilise des données stockées dans une base de données qui est gérée par un Système de Gestion de Bases de Données (SGBD).



# Outils de développement d'applications

Il existe différents outils de développement d'applications de BD :

- Langages de programmation à usage général : (ex. COBOL, C, C++, JAVA, ...)
- Langages de programmation intégrés dans la BD (ex. PL/SQL d'Oracle, ..)
- Générateurs d'applications : (exemple Developer)
  - Outils permettant de développer des applications, principalement, à travers des spécifications *déclaratives* à partir d'une interface graphique.



# Conception des schémas relationnels

# Conception d'un schéma relationnel

- Objectif
  - Comment regrouper des attributs dans des schémas de relations ?
- Problèmes éventuels
  - Mélange d'informations différentes dans un même schéma de relation
  - Éclatement des schémas de relations en des « unités » trop petites
- Une bonne conception
  - Trouver un bon compromis entre l'éclatement des données dans des petites relations et le regroupement des données dans des grandes relations

# Mesures (informelles) pour une bonne conception des schémas relationnels

- 1) Sémantique des attributs
- 2) Réduire (supprimer) les valeurs nulles
- 3) Réduire les valeurs redondantes dans les tuples



# 1) Sémantique des attributs

- Les schémas de relations doivent avoir une sémantique claire

Ex) Expliquez ces deux schémas de relations ?

*Employé(nss, nom, dateN, adresse, Depnum)*

*EMP-DEP(nss, nom1, dateN, adresse, Depnum, nom2, numhabitant)*

Note) Le schéma EMP-DEP mélange des informations concernant des objets différents du monde réel.

## **Principe 1**

*Concevoir des schémas de relations dont l'interprétation est facile*

## 2) Problèmes des valeurs nulles

- Comment les interpréter ?
  - Valeur non applicable
  - Valeur inconnue
  - Valeur absente
- Exemples d'erreurs fréquentes
  - Ex) « Liste des employés qui ne travaillent pas dans le département numéro 3 »
  - Ex) si on veut calculer la moyenne sur un attribut contenant NULL ?
- La seule opération permise sur les valeurs NULL est une opération de test ; IS NULL, IS NOT NULL.
  - Ex) `SELECT nss FROM Empolyé WHERE Depnum IS NULL;`

# Tests sur les valeurs nulles

<b>AND</b>	<b>Null</b>
<b>Vrai</b>	Null
<b>Faux</b>	Faux
<b>Null</b>	Null

<b>OR</b>	<b>Null</b>
<b>Vrai</b>	Vrai
<b>Faux</b>	Null
<b>Null</b>	Null

<b>Not</b>	<b>Null</b>
<b>Null</b>	Null

	<b>IS Null</b>	<b>IS not Null</b>
<b>Vrai</b>	Faux	Vrai
<b>Faux</b>	Faux	Vrai
<b>Null</b>	Vrai	Faux

## Principe 2

*Dans la mesure du possible, éviter d'avoir dans une relation des attributs qui peuvent avoir des valeurs nulles. Sinon, assurez-vous que les valeurs nulles sont des cas exceptionnels (ne s'appliquent pas à la majorité des tuples de la relation).*

Note) Contrainte d'intégrité de valeur obligatoire  
=> NOT NULL

### 3) Attention à la redondance d'information

- Objectif important de la conception :  
Réduire l'espace de stockage
  - Comparer l'espace utilisé pour le stockage des informations pour les deux cas suivants:

Cas 1)

EMP-DEP(nss, nom, dateN, adresse, Depnum, dnom, numhabitant, ..)

Cas 2)

Employé(nss, nom, dateN, adresse, Depnum),  
Département(Depnum, dnom, numhabitant, ...)

- *La redondance est la source de beaucoup de problèmes*
  - *Stockage redondant*
  - *Anomalies de mise à jour : risque d'incohérence des données*
  - *Perte d'information*

Produit	Client	Adresse	Quantité commandée	Montant
lotion	Martin	Paris	10	250
laque	Martin	Paris	250	2500
creme	Martin	Paris	20	1000
lotion	John	Londres	30	750
creme	Dupont	Lyon	20	1000
lotion	Dubois	Paris	20	500

- Clé : (Client, Produit)
- Relation correcte pour un SGBD mais qui va engendrer de nombreux problèmes de maintenance et des anomalies.
- Stockage redondant
  - *Adresse* du client répétée autant de fois que de produits commandés
- Anomalies de mise à jour
  - mise à jour *Adresse* nécessite plusieurs modifications
  - anomalies car plusieurs adresses pour le même client
    - Ex) insertion <creme, John, New York, 10, 500>
- Perte d'information
  - Si un client annule ses commandes, alors on perd son nom et son adresse.

## **Principe 3**

*Concevoir des schémas de relations  
minimisant les redondances pour  
éviter les anomalies.*

# Comment éviter les anomalies ?

- Les contraintes d'intégrité, particulièrement les dépendances fonctionnelles, peuvent être utilisées pour identifier les schémas avec ce type de problèmes et de suggérer de les affiner.
- Technique principale de raffinement :  
Décomposition (par exemple, remplacer un schéma de relation ABCD par AB et BCD, ou ACD et ABD).
- La décomposition doit être utilisée avec précaution :
  - Y-a-t-il des raisons pour décomposer une relation ?
  - Quels problèmes (éventuels) seront engendrés par la décomposition ?

# Décomposition d'un schéma de relation

- Considérons un schéma de relation  $R(A_1 \dots A_n)$ .  
*Une décomposition de R* consiste à remplacer R par deux ou plusieurs schémas de relation tels que :
  - Chaque nouveau schéma de relation contient un sous-ensemble des attributs de R (et aucun attribut qui n'appartient pas à R),
  - Chaque attribut de R apparaît comme un attribut d'une des nouvelles relations
- On stocke dans la base de données les instances des schémas issues de la décomposition de R au lieu des instances de R

## Exemple

Décomposition de la relation **EMP-DEP** :

*EMP-DEP(nss, nom, dateN, adresse, Depnum, dnom,  
numhabitant)*

=>

*Employé(nss, nom, dateN, adresse, Depnum)*

*Département(Depnum, dnom, numhabitant)*



# Problèmes engendrés par la décomposition

Trois problèmes potentiels :

- Certaines requêtes peuvent être plus coûteuses
  - Nom du département dans lequel travaille l'employé ' Dupont '
- Vérifier certaines contraintes peut nécessiter de faire des jointures entre les instances des relations décomposées
  - Ex) contrainte référentielle : tout *Depnum* dans la table *Employé* doit être présent dans la table *Département*.
- Étant donné une instance d'une relation décomposée, on n'est pas capable de retrouver l'instance correspondant à la relation originale.
  - Exemple : décomposition de la relation Works-on :  
**Works-on(empnum, hours, projnumber, projname, responsable)**  
  
=>  
**EMP-PROJ (empnum, hours, projnumber)**  
**PROJ (projname, responsable)**

# Propriété de la décomposition

- Décomposition sans perte d'information :  
La décomposition doit être réversible pour permettre de retrouver l'instance de la relation originale.
  - La relation initiale est obtenue par jointure des composants.

**Ex) Works-on(empnum, hours, projnumber,  
projname, responsable)**

=>

**EMP-PROJ (empnum, hours, #projnumber)**

**PROJ (projnumber, projname, responsable)**

# Retour sur la conception d'un schéma relationnel

- Une autre formulation du problème de conception :
  - Trouver un compromis entre les problèmes engendrés par la décomposition des schémas de relations et la redondance
    - Comment déterminer si un schéma relationnel a besoin d'être décomposé ?
- Pour des applications complexes, le « bon sens » n'est pas suffisant.
- Théorie de la normalisation
  - Représentation formelle et précise du « bon sens »
  - Basée sur la notion de dépendances fonctionnelles.

# Dépendance fonctionnelles (DF)

## (rappel)

- Il y a une dépendance fonctionnelle entre deux attributs ou groupe d'attributs A et B, si la connaissance d'une valeur de A permet de déterminer **une** valeur unique pour B.
- Notation :  $A \rightarrow B$ 
  - A détermine B ou B dépend fonctionnellement de A
  - A est la source de la DF, B en est le but
- *Exemple de DF*  
*EMP-DEP : nss  $\rightarrow$  ename*

# A quoi sert la normalisation ?

- Si une relation est dans une certaine forme normale (FNBC, 3FN etc.), on sait que certains types de problèmes sont évités/minimisés. Ceci peut nous aider à décider si une relation doit être décomposée.
- Rôle des DF : détection de la redondance.
- La seule manière de déterminer une DF est de regarder soigneusement ce que signifient les attributs.
- Considérons une relation R ayant 3 attributs, ABC.
  - Si pas de DF : Pas de redondance
  - Si  $A \rightarrow B$  : Plusieurs tuples peuvent avoir une même valeur pour A, et donc ils auront une même valeur pour B
  - Ex) (client, adresse, articlecommandé)  
DF : client  $\rightarrow$  adresse

# Formalisation de la clé d'une relation

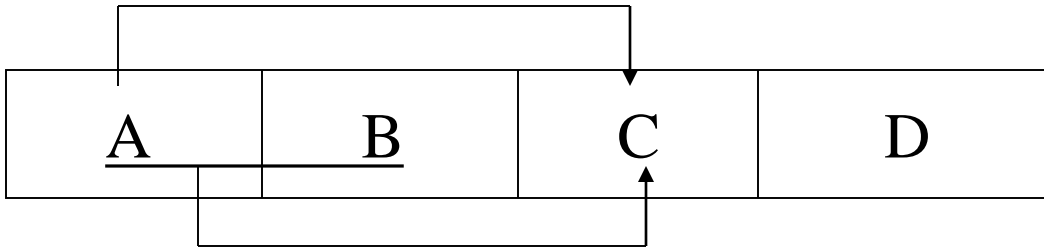
Lorsque, dans une relation, un attribut (ou un groupe d'attributs) est source de dépendances fonctionnelles ayant respectivement pour but chacun des autres attributs de la relation, cet attribut (ou ce groupe d'attributs) est appelé clé de la relation.

## DF élémentaire et directe

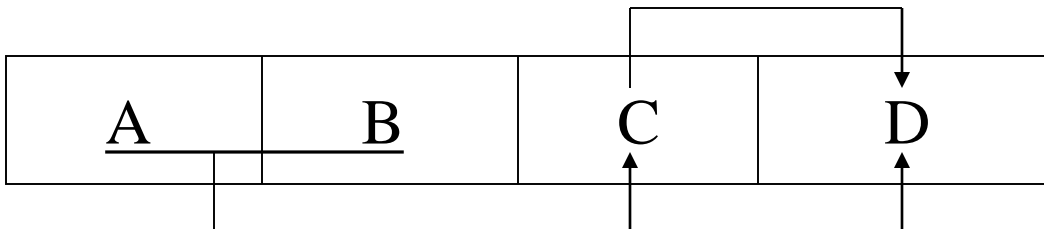
- On dit qu'une DF est élémentaire si aucun sous-ensemble de la clé n'est source de la dépendance fonctionnelle.
  - $A, B, C \rightarrow D$  est élémentaire si on n'a pas :
    - $A, B \rightarrow D$  et  $A, C \rightarrow D$  et  $B, C \rightarrow D$  et
    - $A \rightarrow D$  et  $B \rightarrow D$  et  $C \rightarrow D$
- $A \rightarrow B$  est directe s'il n'existe pas un autre attribut  $C$  tel que  $A \rightarrow C$  et  $C \rightarrow B$ .

# Formes normales (rappel)

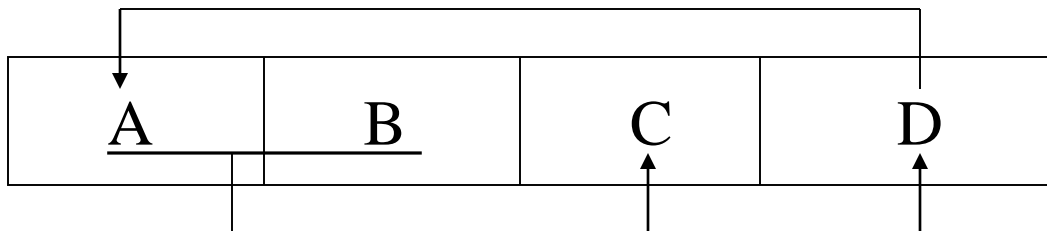
Non 2 FN



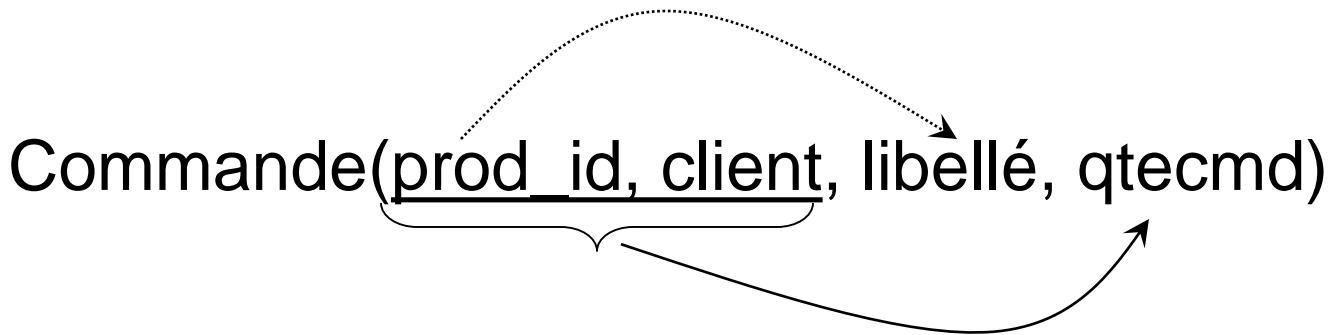
Non 3 FN



Non FNBC(FN de Boyce-Codd)



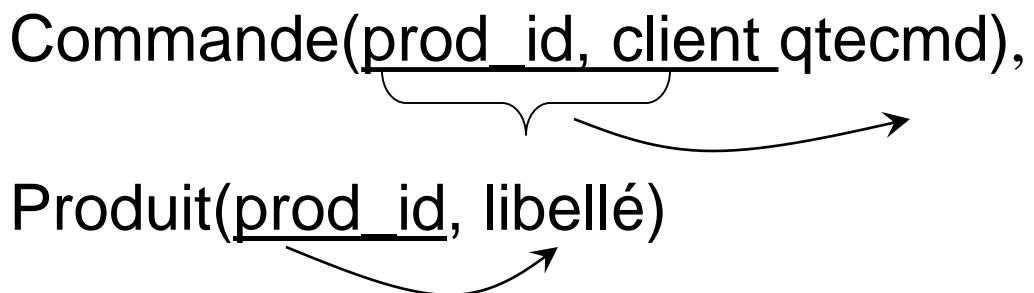
## Ex) 1 FN et Non 2 FN



- Problèmes :

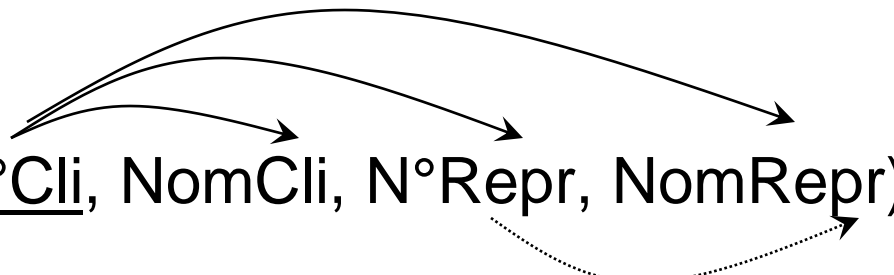
- Seule DF (prod\_id, client) → qtecnd est prise en compte
- On peut donc introduire des anomalies comme :
  - <23, Martin, Creme, 200>
  - <23, Dupont, Lotion, 100>

### *Décomposition* ➔

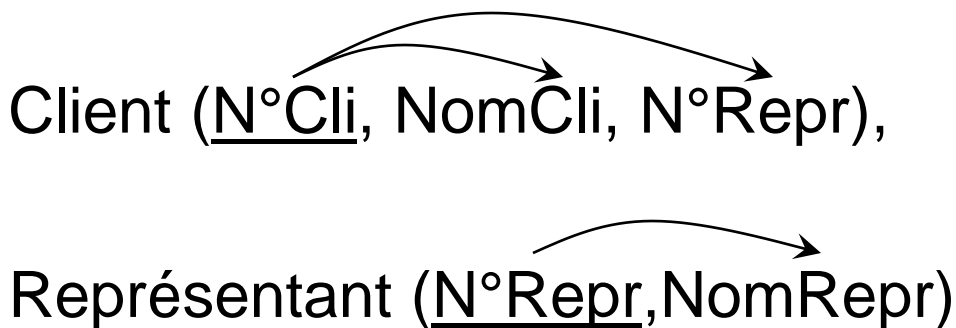




## Ex) 2 FN et Non 3 FN

- Client (N°Cli, NomCli, N°Repr, NomRepr)
- Problèmes :
  - La DF N°Repr → NomRepr n'est pas prise en compte
  - On peut donc introduire des anomalies comme :  
<92, MARLENE, R15, LOULOU>  
<156, PHILIPPE, R15, BABETTE>

*Décomposition* ➔

Client (N°Cli, NomCli, N°Repr),  
Représentant (N°Repr, NomRepr)

# Exercice sur la normalisation

- Trouvez les formes des relations :

R1(N°Client, N°Produit, QtComm, NomProd)

R2(N°Commande, N°Produit, QtComm)

R3(N°Client, NomClient, Nom Repr)

R4(N°Produit, NomProduit, N°Atelier, NomChefAtelier)

R5(N°Client, NomClient, N°Repr, NomRepr)

R6(N°Produit, N°Fournisseur, Prix, NomFourn)

# Discussion

- Un modèle relationnel en FNBC est considéré comme étant de la qualité suffisante pour l'implantation.
  - Une bonne heuristique : assurer que toutes les relations sont en FNBC
  - Les cas de tables modélisées et transformées en 3FN qui ne sont pas déjà en FNBC sont très rares.
  - Attention : parfois il n'est pas possible de décomposer jusqu'à la FNBC sans perte de dépendances. Dans ce cas, il faut faire un choix :
    - Perdre une DF
    - Trouver une décomposition en 3FN
- Dans un SGBD relationnel l'unique façon de prendre en compte une DF est de déclarer sa source comme clé d'une relation.
  - ce qui interdit la duplication des valeurs identiques et assure ainsi la cohérence des valeurs.
- La normalisation n'est pas une loi
  - Il peut y avoir des raisons pour ne pas normaliser.