

# NOMENCLATURE

---

- **Table** = relation = entité
  - **Colonnes** = champs = attributs = propriétés des lignes dans une table
  - **Lignes** = Tuples = objets = enregistrements (données).
  - **Schema** = ensembles des attributs
  - **Contrainte d'intégrité référentielle** = clé étrangère (foreign key)
- 

## Clé primaire (primary key)

Une clé, c'est un groupe d'attributs (= un groupe de colonnes).

Pour qu'un groupe d'attribut soit une clé, il faut être sûr que deux tuples (= deux lignes) n'auront jamais des valeurs identiques pour ces attributs.

Ce groupe d'attributs doit être minimal, c'est-à-dire que si on retire l'un des attributs de ce groupe, la phrase précédente n'est plus vérifiée.

---

## Clé candidate

Quand plusieurs clés sont possibles sur une table, on les appelle des clés candidates. Parmi les clés candidates, on en choisit une qui servira de référence: ce sera alors la clé primaire de cette table.

---

## Clé artificielle

Si aucune clé candidate n'est simple et intelligible, on crée une clé artificielle, souvent appelée identifiant.

**Note:** on utilise parfois les clés artificielles dans les SGBDR pour des questions de performance

---

## Clé étrangère (foreign key)

Une clé étrangère sert à lier des relations (= des tables) entre elles.

On dit qu'une clé étrangère d'une table A référence la clé primaire d'une table B.

---

## Redondance de données

Désigne le fait de répéter une information sous plusieurs formes sans que cela soit nécessaire

Si une table contient de la redondance, mieux vaut la séparer en plusieurs tables avant de la stocker dans la base de données.

Il y a une règle pour savoir comment séparer une table redondante, on appelle cette règle "normalisation".

# ALGEBRE RELATIONNELLE

**def:** La représentation d'information sous forme relationnelle est intéressante car les fondements mathématiques du relationnel, outre qu'ils permettent une modélisation logique simple et puissante, fournissent également un ensemble de concepts pour manipuler formellement l'information ainsi modélisée.

Ainsi une algèbre relationnelle, sous forme d'un ensemble d'opérations formelles, permet d'exprimer des questions, ou requêtes, posées à une représentation relationnelle, sous forme d'expressions algébriques.

L'algèbre relationnelle est composée par les cinq opérateurs de base et les trois opérateurs additionnels suivants :

\*\* opérateurs de base \*\*

- Union
- Différence
- Projection
- Restriction
- Produit cartésien

\*\* opérateurs additionnels \*\*

- intersection
- jointure
- division

## PROJECTION

**def:** La projection est une opération unaire (c'est à dire portant sur une seule relation). La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

Note: Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

## exemple

Soit la relation suivante :

Personne (#Nom, Prénom, Age)		
Dupont	Pierre	20
Durand	Jean	30

Soit l'opération suivante :

R = Projection (Personne, Nom, Age)		

On obtient alors la relation R composée des tuples suivants :

R	
Dupont	20
Durand	30

## RESTRICTION

**def:** La restriction est une opération unaire (c'est à dire portant sur une seule relation). La restriction de R1, étant donnée une condition C, produit une relation R2 de même schéma que R1 et dont les tuples sont les tuples de R1 vérifiant la condition C.

## exemple

Soit la relation suivante :

Personne (#Nom, Prénom, Age)		
Dupont	Pierre	20
Durand	Jean	30

Soit les tuples suivants :

Personne		
Dupont	Pierre	20
Durand	Jean	30

Soit l'opération suivante :

R = Restriction (Personne, Age>25)		

On obtient alors la relation R composée de l'unique tuple restant suivant :

R		
Durand	Jean	30

## Produit cartésien

**def:** Le produit cartésien est une opération binaire (c'est à dire portant sur deux relations). Le produit de R1 par R2 (équivalent au produit de R2 par R1) produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble des combinaisons possibles entre les tuples de R1 et ceux de R2.

Note: Le nombre de tuples résultant du produit de R1 par R2 est égal au nombre de tuples de R1 fois le nombre de tuples de R2.

Note: Le nombre de colonne du produit de R1 par R2 est égal au nombre de colonne de R1 plus le nombre de colonnes de R2.

## exemple

Soit les deux relations suivantes :

Homme (#Nom, Prénom, Age)		
Dupont	Pierre	20
Durand	Jean	30

Soit les tuples suivants pour ces deux relations respectivement :

Homme		
Dupont	Pierre	20
Durand	Jean	30

  

Voiture		
Tesla	Model X	Dupont
Citroën	2 CV	Durand

Soit l'opération suivante :

R = Produit (Homme, Voiture)		

On obtient alors la relation R composée des tuples suivants :

R					
Dupont	Pierre	20	Tesla	Model X	Dupont
Dupont	Pierre	20	Citroën	2 CV	Durand
Durand	Jean	30	Tesla	Model X	
Durand	Jean	30	Citroën	2 CV	Durand

## Jointure

**def:** La jointure est une opération binaire (c'est à dire portant sur deux relations). La jointure de R1 et R2, étant donné une condition C portant sur des attributs de R1 et de R2, de même domaine, produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble de ceux obtenus par concaténation des tuples de R1 et de R2, et qui vérifient la condition C.

## exemple

Soit les deux relations suivantes :

Homme (#Nom, Prénom, Age)		
Dupont	Pierre	20
Durand	Jean	30

Soit les tuples suivants pour ces deux relations respectivement :

Homme		
Dupont	Pierre	20
Durand	Jean	30

  

Voiture		
Tesla	Model X	Dupont
Citroën	2 CV	Durand

Soit l'opération suivante :

R = Jointure (Homme, Voiture, Homme.Nom=Voiture.Propriétaire)		

On obtient alors la relation R composée des tuples suivants :

R					
Dupont	Pierre	20	Tesla	Model X	Dupont
Dupont	Pierre	20	Citroën	2 CV	Durand

1 (Dupont, Pierre, 20, Dupont, Georges, 1)  
2 (Dupont, Pierre, 20, Dupont, Jacques, 3)

### Jointure naturelle

**def:** La jointure naturelle entre R1 et R2 est une jointure pour laquelle la condition est l'égalité entre les attributs de même nom de R1 et de R2. Il est donc inutile de spécifier la condition dans une jointure naturelle, elle reste toujours implicite.

## exemple

Soit les deux relations suivantes :

Homme (#Nom, Prénom, Age)		
---------------------------	--	--

# CONTRAINTE D'INTEGRITE

---

## Definition

Quel que soit le modèle de données (entité association, relationnel ou autre), il existe toujours des règles du monde réel qui ne peuvent pas être exprimées par les concepts du modèle. Certaines de ces règles restreignent les valeurs que peuvent prendre les données de la base. Elles sont appelées Contraintes d'intégrité.

---

## Differents types de contrainte d'integritE

- **Contraintes de domaine**

Qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table

- **Contraintes d'integrite d'entite**

Qui précisent qu'une table doit toujours avoir une clé primaire

- **Contraintes d'integrite referentielle**

Qui précisent les conditions dans lesquelles peuvent être ajoutées ou supprimées des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères

- **Contraintes d'integrite quelconque**

Qui permet de spécifier que lors de toute insertion (ou suppression ou modification) d'un tuple dans telle relation telle condition doit être satisfaite sinon telle action doit être entreprise automatiquement par le SGBD, comme par exemple refuser l'insertion ou envoyer un message d'alerte.

---

## Exemples pour chaque type de contrainte d'integritE

- **Contraintes de domaine**

Domaine de valeurs particulier d'un attribut: clause CHECK

- **Contraintes d'integrite d'entite**

Identifiant: clauses PRIMARY KEY et UNIQUE Attribut obligatoire: clause NOT NULL

- **Contraintes d'integrite referentielle**

Clause FOREIGN KEY

- **contrainte d intégrité quelconque**

Clause TRIGGER

# DEPENDANCE FONCTIONNELLE

## Definition

Mathématiquement, on dit que X détermine Y, ou que Y dépend fonctionnellement de X, si et seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y, on note  $X \rightarrow Y$ .

## Exemple de DF

Soit la relation Eleve(Matricule, Nom, Age, Sexe, Province, Ville)

```
Matricule → Nom, Age, Sexe, Province, Ville  
Ville → Province
```

## Utilité

La notion de DF a été introduite dès le début du relationnel par Codd afin de caractériser des relations décomposables sans perte d'information.

## Signification (exemple)

Annee, Etudiant->Groupe: qui indique que chaque année, un étudiant appartient à un seul groupe

Groupe, Matiere->Professeur: qui indique qu'un seul professeur est affecté à un groupe

Professeur->Matiere: qui indique qu'un professeur n'enseigne qu'une seule matière

## Démonstration

Pour montrer la non DF, il suffit de citer un contre exemple en posant la question suivante: "Est-ce que 2 valeurs différentes de Y peuvent-elles être associées à une seule valeur de X?"

```
- Si la réponse est "NON" → il y a DF  
- Si la réponse est "OUI" → il n'y a pas de DF
```

## Exemple de demo

Soit la relation:

```
Etudiant(Matricule, Nom, Classe, NumeroDeClasse, Annee, Matiere, DateDS, Note)
```

## Donnez les DF et les non DF de cette relation

Nous savons qu'un matricule est un numéro unique à chaque étudiant lorsqu'il entre dans un établissement et que ce numéro ne sera plus jamais attribué à un autre étudiant.

### • DF

Par conséquent, il peut y avoir deux étudiants différents portant le même nom, dans ce cas, il y aura 2 numéros d'inscription différents pour la même valeur de Nom. Autrement dit, le Nom n'est pas déterminé par Matricule mais l'inverse est possible et que l'on note : Matricule-Nom. De même, pour un élève, une matière et une date de DS donnés, il y a une et une seule note. Ce qui donne le DF: (Matricule, Matiere, DateDS) → Note

### • Non DF

Considérons maintenant les attributs Matricule et Classe : Durant l'année académique 2011, l'étudiant avec le matricule 110001 (par exemple) est en classe de 1ère année mais durant l'année 2012, ce même étudiant est en 2ème année pour la même valeur de matricule, on peut alors avoir 2 valeurs de classe différentes. Nous concluons que "Matricule ne détermine pas la classe" Matricule !→ Classe. Par inverse, il est évident que Classe !→ Matricule car il y a beaucoup d'élèves dans la même classe. En revanche, si l'on considère le couple d'attributs (Matriculation, Année), il est désormais possible de déterminer la classe d'un étudiant au cours d'une année académique par la DF: (Matriculation, Année) → Class

## Types de DF

### • DFE (Dépendance Fonctionnelle Élementaire)

Une DF  $X \rightarrow Y$  est élémentaire si et seulement si on enlève une des attributs de la source X et que le reste ne détermine plus Y. Autrement dit, si Y est pleinement dépendant de l'intégralité de X Ex: (Matricule, Année) → Classe

**Note:** Toute DF dont la source X est un seul attribut est alors DFE car si on enlève la source, il n'y aura plus que le vide. Donc plus de DF

### • DFD (Dépendance Fonctionnelle Directe)

Une DF  $X \rightarrow Y$  est DFD si et seulement si,  $X \rightarrow Z$  et  $Z \rightarrow Y$ . On dit alors que Y est directement dépendant de X

### • DFPGC (Dépendance Fonctionnelle à Partie Gauche Composée)

Une DF  $X \rightarrow Y$  est DFPGC lorsque X est formée par plusieurs attributs Ex: (Matricule, Matiere, DateDs) → Note

## Propriétés de DF (Axiomes d'Amstrong)

### • Reflexivité

Tout groupe d'attributs se détermine lui-même et détermine chacun de ses attributs (ou sous groupe de ses attributs)

Soient X et Y des attributs:  $XY \rightarrow XY$  et  $XY \rightarrow X$  et  $XY \rightarrow Y$

### • Augmentation

Si un attribut X détermine un attribut Y, alors tout groupe composé de X enrichi avec d'autres attributs détermine un groupe composé de Y et enrichi des mêmes autres attributs.

Soient X, Y et Z des attributs: Si  $X \rightarrow Y$  alors  $XZ \rightarrow YZ$

### • 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: Si  $X \rightarrow Y$  et  $Y \rightarrow Z$ , alors  $X \rightarrow Z$

### • Pseudo-transitivité

Si un attribut X détermine un autre attribut Y, et que Y appartient à un groupe G qui détermine un troisième attribut Z, alors le groupe G obtenu en substituant Y par X dans G détermine également Z.

Soient W, X, Y et Z des attributs:  $X \rightarrow Y$  et  $WY \rightarrow Z$  alors  $WX \rightarrow Z$

**Note:** Cette propriété est déduite de l'augmentation et de la réflexivité  $X \rightarrow Y$  et  $WY \rightarrow Z \Rightarrow WX \rightarrow WY$  et  $WY \rightarrow Z \Rightarrow WX \rightarrow Z$

### • Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs.

Soient X, Y et Z des attributs : Si  $X \rightarrow Y$  et  $X \rightarrow Z$  alors  $X \rightarrow YZ$

**Note:** Cette propriété est déduite de la réflexivité et de l'augmentation  $X \rightarrow Y$  et  $X \rightarrow Z \Rightarrow X \rightarrow XX$  et  $XX \rightarrow XY$  et  $XY \rightarrow YZ \Rightarrow X \rightarrow YZ$

### • Décomposition (inverse de l'union)

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 : Si  $X \rightarrow YZ$  alors  $X \rightarrow Y$  et  $X \rightarrow Z$

**Note:** Cette propriété est déduite de la réflexivité et de la transitivité  $X \rightarrow YZ \Rightarrow X \rightarrow YZ$  et  $YZ \rightarrow Z \Rightarrow X \rightarrow Z$

# NORMALISATION

## Definition

La normalisation est un processus de décomposition d'une table universelle en plusieurs tables en évitant le problème d'incohérence et de redondance sans perdre d'information tout en préservant les DFs

## Roles de la normalisation

- Limiter la redondance des données
- Limiter la perte de données
- Limiter les incohérences de données
- Améliorer les performances de traitement

## Les inconvenients d'une BDD non normalisée

- Risque d'avoir des données redondantes et des tables volumineuses
- Risque d'avoir des incohérences de données
- Nuire au performances de traitement
- Difficulté lors du maintenances de la base de données

## Les 3 premières formes normales

1FN, 2FN, 3FN (*mais il y a aussi la forme normale BCNF de Boyce Codd*)

Note: il existe 6 formes normales en total

### Relations entre les 3 premières formes normales + BCNF

#### • 1FN

Une relation est 1FN si et seulement si tous ses attributs ont des valeurs atomiques (non multiples).

=> Par définition d'une relation, toute relation est donc en 1FN

attribut.s non atomique.s (separation possible)

#### Produit

pc-0001

cell

samsung

ord-03a

pc

dell

mon-99

montre

casio

attribut.s atomique.s

#### id produit   type produit   modèle produit

pc-0001   cell   samsung

ord-03a   pc   dell

mon-99   montre   casio

#### • 2FN

Une relation est en 2FN si et seulement si:

1. La relation est en 1FN
2. Tout attribut non clé est pleinement dépendant de l'intégralité de la clé ou bien toute DF entre la clé et un attribut non clé est élémentaire. Autrement dit tout attribut non clé ne dépend pas d'une partie de la clé. Donc décomposition possible en plusieurs relations ( $R \rightarrow R_1, R_2, \dots, R_n$ ). La 2FN élimine donc les anomalies résultant des dépendances entre partie de clé et partie non clé.

=> Toute relation dont la seule clé est un attribut est une relation en 2FN

#### Exemple de non 2FN d'une relation

Commande(codeClient, codeArticle, client, article) n'est pas en 2FN car la clé de cette relation est (codeClient, codeArticle) alors qu'on a aussi des DF `codeClient->client` et `codeArticle->article`

#### Exemple d'une relation en 2FN

Etudiant(NumBacc, AnnéeBacc, Nom, Prenom, Filière) est en 2FN car sa clé est (NumBacc, AnnéeBacc) et aucune DF ne peut pas partir sur l'un de ces clés. Il faut l'intégralité de l'attribut clé pour déterminer les attributs non clés

#### • 3FN

Une relation est en 3FN si et seulement si:

1. La relation est en 2FN
2. Toutes les DF entre attribut clé et attribut non clé sont directes, c'est à dire pas de DF transitive. Autrement dit tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé.

#### Exemple de non 3FN d'une relation

Etudiant(Matricule, Nom, Classe, EffectifClasse) n'est pas en 3FN. En effet, `Matricule->Nom, Classe, EffectifClasse`. Or, on a aussi `Classe->EffectifClasse`. Donc on a une DF transitive: `Matricule->Classe->EffectifClasse` cause du non 3FN de cette relation

#### 3FN pour la relation Etudiant

On peut alors décomposer la relation comme suit pour avoir une relation en 3FN: `Etudiant(Matricule, Nom, CodeClasse)` `Classe(CodeClasse, EffectifClasse)`

#### • BCNF

Une relation est en BCNF (Forme normale de Boyce Codd) si et seulement si:

1. La relation est en 3FN
2. Toutes les DF qui existent sont issues d'une seule clé primaire

#### Exemple d'une relation en BCNF

Etudiant(Matricule, Nom, Prenom, Adresse, Adresse\_mail, Tel, Sexe, Niveau\_id, Ville, Region). Tous les attributs non clés sont déterminés par la clé primaire 'matricule'. Mais la cause du non BCNF c'est la DF: `Ville->Region` car Ville n'est pas une clé => Par décomposition on obtient alors une autre table `Ville(Ville_id, Region)` ce qui respecte la condition du BCNF

# DENORMALISATION

## Definition

La dénormalisation est un processus consistant à regrouper plusieurs tables liées par des références, en une seule table, en réalisant les opérations de jointure adéquates.

## Objectif

L'objectif de la dénormalisation est d'améliorer les performances de la BD en implémentant les jointures plutôt qu'en les calculant.

## Interet (cas d'utilisation)

Un schéma doit être dénormalisé lorsque les performances de certaines recherches sont insuffisantes et que cette insuffisance est due à la présence de jointures.

## Inconvénients

La dénormalisation peut également avoir un effet néfaste sur les performances si on ne contrôle pas la redondance volontaire:

#### • En mise à jour

Les données redondantes doivent être dupliquées plusieurs fois.

#### • En contrôle supplémentaire

Les moyens de contrôle ajoutés (triggers, niveaux applicatifs, etc.) peuvent être très coûteux.

#### • En recherche ciblée

Certaines recherches portant avant la normalisation sur une "petite" table et portant après sur une "grande" table peuvent être moins performantes après qu'avant.

## DIFFERENCE ENTRE NORMALISATION & DENORMALISATION

### Normalisation      Denormalisation

La normalisation consiste à diviser des tables plus volumineuses en tables plus petites, réduisant ainsi les données redondantes.

La dénormalisation consiste à ajouter des données redondantes afin d'optimiser les performances.

La normalisation est effectuée pour éviter les anomalies des bases de données.

Une base de données dénormalisée peut offrir de meilleures performances en écriture qu'une base de données normalisée.

# BD REPARTIE

Differences entre BD repartie et BD parallèle

- **BD parallèles**

Les données peuvent être réparties sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

- **BD réparties**

Les données sont diffusées et/ou dupliquées sur différents sites du réseau (ex: internet) qui disposent d'une certaine degré d'autonomie. Chaque site peut avoir une BD parallèle.

Les Avantages d'une BD repartie par rapport à une architecture centralisée

- **Performance**

En rapprochant les données des applications utilisant ces données (ex: stockant les comptes des clients montréalais dans un site localisé à Montréal), on peut réduire les coûts de transfert sur le réseau et, ainsi, augmenter la performance des requêtes sur ces données.

- **Fiabilité**

En dupliquant certaines données importantes sur plusieurs sites, on minimise l'impact d'une panne sur un site. De même, en cas de panne, on peut rediriger le traitement d'une requête vers un autre site disponible.

- **Extensibilité**

Si les besoins en espace de stockage et en puissance de traitement augmentent on peut facilement rajouter un nouveau nœud (site), sans avoir à remplacer le serveur (ex : approche Google).

Le principe de la stratégie d'optimisation par semi-jointure

La stratégie par semi-jointure permet de réduire le coût d'une jointure en limitant la quantité de données transférées sur le réseau.

Supposons que l'on veuille calculer  $T1 \bowtie T2$  où la table  $T_i$  est située sur le site  $i$ . Au lieu de transférer une table complète d'un site à un autre, on envoie seulement les colonnes nécessaires à la jointure (la clé). Par exemple, on envoie  $\pi_{\text{clé}}(T2)$  au site 1 et on fait la jointure avec  $T1$ :

$$R = T1 \bowtie \pi_{\text{clé}}(T2)$$

Ceci correspond à faire la semi-jointure entre  $T1$  et  $T2$ . Ensuite, on envoie le résultat  $R$  au site 2 pour faire la jointure avec  $T2$ :

$$T = R \bowtie T2 = T1 \bowtie T2$$

Les données transférées sont celles de  $\pi_{\text{clé}}(T2)$  et de  $R$ , et ont une taille potentiellement moins grande que celle de  $T1$  ou de  $T2$ .

Role de la répartition cyclique par bloc employée dans certaines architectures RAID

Au lieu de disposer les blocs d'une table séquentiellement sur un même disque, la répartition cyclique les dispose en alternance sur plusieurs disques. Par exemple, Disque 1 Disque 2 Disque 3 bloc 1 bloc 2 bloc 3 bloc 4 bloc 5 bloc 6 ... .... Le but de cette stratégie est de permettre la lecture / écriture de plusieurs blocs en parallèle (un dans chaque disque).

La différence entre la fragmentation horizontale et verticale d'une table

- **Fragmentation horizontale**

Chaque fragment contient un sous-ensemble de lignes de la table. Par exemple, on découpe la table Client selon la provenance (ex : province, état, etc.) d'un client.

- **Fragmentation verticale**

Chaque fragment contient un sous-ensemble de colonnes de la table. En pratique, ce type de fragmentation est rarement employé.

Avantages de la fragmentation dans le contexte des bases de données réparties

• La fragmentation horizontale permet de répartir les lignes d'une table sur les sites où le traitement de ces lignes est souvent fait, réduisant ainsi les temps de transfert sur le réseau.

• En cas de panne d'un site, l'information stockée sur les autres sites reste disponible.

Avantages de la duplication dans les BD réparties

• Réduit les coûts de transfert en dupliquant sur les différentes l'information globale à tous les sites. Par exemple, les codes et les frais associés aux transactions bancaires.

• Assure la disponibilité des données dupliquées dans le cas où un ou plusieurs sites tombent en panne.

La différence entre la duplication répartie synchrone et asynchrone

- **Duplication synchrone**

Une transaction modifiant des données de plusieurs sites n'est confirmée qu'au moment où tous les sites ont confirmés les changements.

- **Duplication asynchrone**

Les mises à jour sont d'abord faites sur la copie primaire des tables, et les autres copies sont mises à jour en différé.

le rôle des vues matérialisées (MATERIALIZED VIEW) dans les BD réparties

Permet de créer une copie locale d'une table située sur un autre site distant. En somme, elles permettent d'implémenter le concept de la duplication (synchrone ou asynchrone).

Différences entre l'optimisation de requêtes dans les BD centralisées et dans les BD réparties

• Coût de communication: Contrairement aux BD centralisées, l'optimisation de requêtes utilisant des données sur plusieurs sites doit également tenir compte du coût de transfert sur le réseau.

• Ressources multiples: L'optimiseur doit également tenir compte de la localisation des données et des diverses ressources à sa disposition. Par exemple, plusieurs sites peuvent contribuer en parallèle à répondre à la requête selon les données qu'ils renferment.

La différence entre les architectures RAID 1 et RAID 5. Dites comment ces architectures se comparent en termes de fiabilité et de performance

- **RAID 1**

Le niveau RAID 1 est basé sur la duplication des données sur des disques miroirs. Cette architecture est robuste aux pannes survenant sur un ou plusieurs disques. De plus, elle permet la lecture en parallèle sur les différents disques (mais pas l'écriture). Par contre, cette architecture est plutôt gourmande en termes d'espace.

- **RAID 5**

Contrairement au niveau RAID 1, le niveau RAID 5 ne duplique pas les données. En revanche, ce niveau emploie la répartition cyclique par bloc ce qui permet de faire des lectures ET des écritures en parallèle. Par ailleurs, elle permet une certaine forme de fiabilité à l'aide de bits de parité stockés séparément des données.

Comment l'opération de sélection peut être accélérée dans les BD parallèles

En supposant que la table sur laquelle opère la sélection est fragmentée, on peut effectuer en parallèle une recherche sur chacun des fragments et ensuite combiner les résultats de ces recherches. Par ailleurs, si la fragmentation est faite selon la clé de sélection, on peut limiter la recherche aux fragments correspondants.

La différence entre les architectures à mémoire partagée et à disque partagés | un avantage et un inconvénient pour chacune d'elles

- **Mémoire partagée**

Plusieurs processeurs (CPU) partagent la même mémoire vive (RAM). L'avantage est que les processeurs peuvent communiquer efficacement à travers la mémoire RAM. Cependant, la mémoire RAM constitue un goulot d'étranglement qui limite le nombre de CPU possibles.

- **Disques partagés**

Contrairement à la précédente, les CPU de cette architecture ont chacun leur propre RAM. Cela facilite l'extension de l'architecture (ajout de nouveau CPU) mais complexifie un peu la communication entre les processeurs. En pratique, cette architecture est celle employée le plus souvent.

Précaution pour sauver et/ou résister les données lors d'une catastrophe

À mon avis, la meilleure façon de sauvegarder et de restaurer des données et d'éviter l'utilisation d'une architecture centralisée en adoptant une base de données distribuée et de dupliquer les données sensibles sur plusieurs bases de données des sites différents. De ce fait, le traitement d'une requête peut être redirigé vers un autre site disponible lors d'une catastrophe.

# BDD AVANCEE - 2019(1)

Soit les relations suivantes:

```
Emprunt(#Personne, #Livre, #DateEmprunt, DateRetourPrevue, DateRetourEffective)
Retard(#Personne, #Livre, #DateEmprunt, PenaliteRetard)
```

## 1. Les personnes ayant emprunté le livre "Recueil Examens BD"

```
R1 = RESTRICT(Emprunt, Emprunt.Livre = "Recueil Examens BD")
R2 = RESTRICT(R1, R1.DateEmprunt != NULL)
R = PROJECT(R2, R2.Personne)
```

## 2. Les personnes n'ayant jamais rendu de livre en retard

```
R1 = RESTRICT(Emprunt, Emprunt.DateRetourEffective < Emprunt.DateRetourPrevue)
R2 = RESTRICT(Emprunt, Emprunt.DateEmprunt != NULL)
R3 = JOIN(R1, R2, R1.Personne = R2.Personne)
R = PROJECT(R3, R3.Personne)
```

## 3. Les personnes ayant emprunté tous les livres (empruntés au moins une fois)

```
R1 = RESTRICT(Emprunt, Emprunt.DateEmprunt != NULL)
R2 = RESTRICT(Emprunt, COUNT(Emprunt.Livre) = COUNT(R1.Livre))
R = PROJECT(R2, R2.Personne)
```

## 4. Les livres ayant été empruntés par tout le monde (de tous les emprunteurs)

```
R1 = RESTRICT(Emprunt, Emprunt.DateEmprunt != NULL)
R2 = RESTRICT(Emprunt, COUNT(Emprunt.Personne) = COUNT(R1.Livre))
R = PROJECT(R2, R2.Livre)
```

## 5. Les personnes ayant toujours rendu en retard les livres qu'elles ont empruntés

```
R1 = RESTRICT(Emprunt, Emprunt.DateEmprunt != NULL)
R2 = RESTRICT(Emprunt, Emprunt.DateRetourEffective > Emprunt.DateRetourPrevue)
R3 = JOIN(R1, R2, R1.Personne = R2.Personne)
R = PROJECT(R3, R3.Personne)
```

# BDD AVANCEE - 2019(2)

Soit les DF suivantes:

Annee, Etudiant->Groupe: qui indique que chaque année, un étudiant appartient à un seul groupe

Groupe, Matiere->Professeur: qui indique qu'un seul professeur est affecté à un groupe

Professeur->Matiere: qui indique qu'un professeur n'enseigne qu'une seule matière

Soit la relation Cours suivante:

Cours(Professeur, Etudiant, Groupe, Matiere, Annee)

## 1. EAR qui caractérise les professeurs ayant travaillé en 2007-2008 pour le groupe CSB6A12

Par décomposition de la relation Cours

GroupeEtu(#Annee, #Etudiant, Groupe)

GroupeProf(#Groupe, #Matiere, Professeur)

MatiereProf(#Professeur, Matiere)

```
R1 = RESTRICT(GroupeEtu, GroupeEtu.Annee = "2007")
R2 = RESTRICT(GroupeEtu, GroupeEtu.Annee = "2008")
R3 = JOIN(R1, R2, R1.Etudiant = R2.Etudiant)
R4 = RESTRICT(GroupeProf, GroupeProf.Groupe = "CSB6A12")
R5 = JOIN(R3, R4, R3.Groupe = R4.Groupe)
R = PROJECT(R5, R5.Professeur)
```

## 2. EAR qui caractérise les professeurs ayant enseigné dans au moins 2 groupes différents pour un même étudiant

Par décomposition de la relation Cours

GroupeEtu(#Annee, #Etudiant, Groupe)

GroupeProf(#Groupe, #Matiere, Professeur)

MatiereProf(#Professeur, Matiere)

```
R1 = PROJECT(GroupeEtu, COUNT(GroupeEtu.Groupe) >= 2)
R2 = JOIN(R1, GroupeProf, R1.Groupe = GroupeProf.Groupe)
R = PROJECT(R2, R2.Professeur)
```

## 3. Cause du non 3FN de la relation Cours

- Pour rappel, une relation est en 3FN si et seulement si:

1. La relation est en 2FN
2. Toutes les DF entre attribut clé et attribut non clé sont directes, c'est à dire pas de DF transitive. Autrement dit tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé.

- Les DF de la relation Cours Annee, Etudiant->Groupe Groupe, Matiere->Professeur Professeur->Matiere

- Si on identifie tous les attributs non clés, on obtient: Groupe, Professeur et Matiere

=> Cependant, on constate qu'il existe une DF entre ces 3 attributs non clés: Groupe fait partie d'une clé pour déterminer Professeur et Professeur permet à son tour de déterminer Matiere. C'est à dire qu'il existe une DF transitive. Ce qui ne respecte pas la condition en 3FN de la relation Cours

# BDD AVANCEE - 2020(1)

## 1. L'importance d'une DF dans une BDD

La notion de DF a été introduite dès le début du relationnel par Codd afin de caractériser des relations décomposables sans perte d'information.

## 2.

DF Total - definition (== DFD != DFE)

Une DF est total lorsqu'un attribut est directement dépendant de son attribut source, autrement dit tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé

Exemple DF total

Matricule->Nom, Prenom, Adresse, Tel

DF transitive - definition

Une DF est transitive lorsqu'une partie des attributs non clés ne dépend pas directement de la clé, c'est à dire que l'une des attributs non clé détermine l'autre attribut non clé

Exemple DF transitive

Matricule->Nom, Prenom, Adresse, Tel, Classe, EffectifClasse Classe->EffectifClasse

## 3. Soit R(A,B,C) une relation, et soit F={AB, AC} l'ensemble de ses DF ou le symbole designe la DF

3.a Vérification de la décomposition de R en R1(A, B) et R2(B, C)

=> La décomposition de R en R1(A, B) est valide car il existe une DF entre l'attribut A et B => La décomposition de R en R2(B, C) n'est pas valide car il n'existe pas une DF entre l'attribut B et C On peut constater que A est alors la clé de cette relation R

3.b Nécessité d'une telle décomposition

## 4. Normalisation

4.a Les inconvénients d'une BDD non normalisée

- Risque d'avoir des données redondantes et des tables volumineuses
- Risque d'avoir des incohérences de données
- Nuire aux performances de traitement
- Difficulté lors de la maintenance de la base de données

4.b Le niveau de normalisation qui devrait satisfaire une BDD

Selon moi, il faut arriver au 3FN pour satisfaire une BDD

4.c Signification du 3FN de Boyce-Codd

3FN de Boyce-Codd signifie que la relation est en 3FN et toutes les DF qui existent sont issues d'une seule clé primaire

Exemple:

Etudiant(Matricule, Nom, Prenom, Adresse, Adresse\_mail, Tel, Sexe, Niveau\_id, Ville, Region) Tous les attributs non clés sont déterminés par la clé primaire 'matricule' Mais la cause du non BCNF c'est la DF: ville->region car ville n'est pas une clé => Par décomposition on obtient alors une autre table Ville(Ville\_id, Region) ce qui respecte la condition du BCNF

## 5.

Soit la relation suivante:

COURS (M: Matière, C: Classe, P: Professeur)

**Règle de gestion:** un professeur n'enseigne qu'une seule matière et une classe n'a qu'un seul enseignant par matière

5.a les DF entre M, C et P

P: Professeur->M: Matière

P: Professeur->C: Classe

5.b M.q cette relation est en 1FN, 2FN, 3FN et FNBC

- Pour 1FN

COURS est une relation donc elle est en 1FN

- Pour 2FN

On sait que la relation COURS est en 1FN et toutes les DF de cette relation sont élémentaires, c'est à dire que si on enlève la source de chaque DF comme Professeur, elle devient vide et on ne détermine plus le reste comme Matière ou Classe. Autrement dit, tout attribut non clé de cette relation est pleinement dépendant de l'intégralité de la clé

- Pour 3FN

On sait que la relation COURS est en 2FN et toutes les DF entre attribut clé Professeur et attribut non clé Matière et Classe sont directes, c'est à dire pas de DF transitive. Autrement dit, tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé

- Pour FNBC

On sait que la relation COURS est en 3FN et toutes les DF qui existent sont issues d'une seule clé primaire qui est Professeur

5.c M.q il est impossible d'enregistrer un professeur sans classe affectée et la disparition d'une classe peut entraîner la disparition du professeur

5.d Solution à ce problème

## 6.a Contrainte d'intégrité référentielle

Definition

Les contraintes d'intégrité référentielle précisent les conditions dans lesquelles peuvent être ajoutées ou supprimées des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères

## 6.c Les anomalies qui peuvent se produire lorsque cette contrainte n'est pas respectée

## 7. Les nouveaux concepts introduits par les BDD parallèles et réparties

- BD parallèles

Les données peuvent être réparties sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

- BD réparties

Les données sont diffusées et/ou dupliquées sur différents sites du réseau (ex: internet) qui disposent d'un certain degré d'autonomie. Chaque site peut avoir une BD parallèle.

# BDD AVANCEE - 2020(2)

## 1.a M.q si X->Y et X->Z alors X->YZ

Soient X, Y et Z des attributs d'une relation. La propriété d'union est déduite de la réflexivité et de l'augmentation dans les axiomes d'Armstrong:

Par réflexivité, tout groupe d'attributs se détermine lui-même et détermine chacun de ses attributs:

Si  $X \rightarrow Y$  et  $X \rightarrow Z$  alors  $X \rightarrow YZ$

Par augmentation, si un attribut X détermine un attribut Y, alors tout groupe composé de X enrichi avec d'autres attributs détermine un groupe composé de Y et enrichi des mêmes autres attributs:

Si  $X \rightarrow Y$  alors  $XY \rightarrow Y$   
Si  $X \rightarrow Z$  alors  $XZ \rightarrow Z$   
Si  $XY \rightarrow Y$ ,  $XZ \rightarrow Z$   
Donc  $X \rightarrow Y, Z$  (cqfd)

## 1.b Application de la propriété à la relation MachineOutil

- Soit la relation suivante:

`MachineOutil(noMachine, marque, debit)`

- Et la DF suivante:

`DF = {noMachine->marque, noMachine->debit}`

On sait que `noMachine` est une clé de la relation `MachineOutil`, d'après la DF donnée, cette clé détermine les attributs non clé `marque` et `debit`, par la règle d'union, on obtient la nouvelle DF suivante:

Si  $noMachine \rightarrow \text{marque}$  et  $noMachine \rightarrow \text{debit}$   
Alors  $noMachine \rightarrow \text{marque}, \text{debit}$

## 1.c L'intérêt pratique de cette propriété

L'intérêt pratique de cette propriété est la `factorisation des plusieurs DF` à condition que chaque attribut non clé partage la même source

## 2.a L'utilité de la normalisation d'une base de données

La normalisation est un processus de décomposition d'une table universelle en plusieurs tables en évitant le problème d'incohérence et de redondance sans perdre d'information tout en préservant les DFs et qui permet de vérifier la robustesse de leur conception pour améliorer la modélisation

## 2.b Cas de dénormalisation d'une relation | inconvénients et nécessité éventuelle d'une dénormalisation

### Intérêt pratique de la dénormalisation

On applique la dénormalisation d'une relation lorsque on a besoin d'améliorer les performances de la BD en implémentant les jointures plutôt qu'en les calculant. Autrement dit, un schéma doit être dénormalisé lorsque les performances de certaines recherches sont insuffisantes et que cette insuffisance à pour cause des jointures

### Inconvénients de la dénormalisation

La dénormalisation peut également avoir un effet néfaste sur les performances si on ne contrôle la redondance volontaire:

- En mise à jour

Les données redondantes doivent être dupliquées plusieurs fois.

- En contrôle supplémentaire

Les moyens de contrôle ajoutés (triggers, niveaux applicatifs, etc.) peuvent être très coûteux.

- En recherche ciblée

Certaines recherches portant avant normalisation sur une "petite" table et portant après sur une "grande" table peuvent être moins performantes après qu'avant.

## 2.c Contre-exemple de 2FN mais non 3FN

### 3.a Normalisation de la table VENTE, COMMANDE et PRODUIT (si nécessaire)

`VENTE(NPRO, CLIENT, DATE, QUANTITE, ADRESSE, DELEGUE, REGION)`

`COMMANDE(NCOM, NCLI, NOM, DATE, NPRO, LIBELLE)`

`PRODUIT(NPRO, DATE_INTRO, IMPORTATEUR, AGREGATION)`

### DF

`NCLI->CLIENT, ADRESSE, DELEGUE, REGION`

`NPRO, NCLI->DATE, QUANTITE`

`NCOM->NCLI, NPRO, NOM, DATE, LIBELLE`

`NPRO->DATE_INTRO, IMPORTATEUR, AGREGATION`

### Normalisation

`CLIENT(NCLI, ADRESSE, DELEGUE, REGION)`

`VENTE(NPRO, NCLI, DATE, QUANTITE)`

`COMMANDE(NCOM, NCLI, NPRO, NOM, DATE, LIBELLE)`

`PRODUIT(NPRO, DATE_INTRO, IMPORTATEUR, AGREGATION)`

## 3.c Type de jointure pour avoir la liste des produits qui n'ont jamais été vendus | explication

### 4.a Définition de la contrainte d'intégrité

Quel que soit le modèle de données (entité association, relationnel ou autre), il existe toujours des règles du monde réel qui ne peuvent pas être exprimées par les concepts du modèle. Certaines de ces règles restreignent les valeurs que peuvent prendre les données de la base. Elles sont appelées Contraintes d'intégrité.

### 4.b Différents types de contrainte d'intégrité

- Contraintes de domaine

Qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table

- Contraintes d'intégrité d'entité

Qui précisent qu'une table doit toujours avoir une clé primaire

- Contraintes d'intégrité référentielle

Qui précisent les conditions dans lesquelles peuvent être ajoutées ou supprimées des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères

- Contraintes d'intégrité quelconque:

Qui permet de spécifier que lors de toute insertion (ou suppression ou modification) d'un tuple dans telle relation telle condition doit être satisfaite sinon telle action doit être entreprise automatiquement par le SGBD, comme par exemple refuser l'insertion ou envoyer un message d'alerte.

### 4.c Exemple pour chaque type de contrainte d'intégrité

- Contraintes de domaine

Domaine de valeurs particulier d'un attribut: clause CHECK

- Contraintes d'intégrité d'entité

Identifiant: clauses PRIMARY KEY et UNIQUE Attribut obligatoire: clause NOT NULL

- Contraintes d'intégrité référentielle

Clause FOREIGN KEY

- Contrainte d'intégrité quelconque

Clause TRIGGER

## 5.a Différences entre BD repartie et BD parallèle | Les Avantages d'une BD repartie par rapport à une architecture centralisée

### Differences

- BD parallèles

Les données peuvent être réparties sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

- BD réparties

Les données sont diffusées et/ou dupliquées sur différents sites du réseau (ex: internet) qui disposent d'une certaine degré d'autonomie. Chaque site peut avoir une BD parallèle.

### Avantages

- Performance

En rapprochant les données des applications utilisant ces données (ex: stockant les comptes des clients montréalais dans un site localisé à Montréal), on peut réduire les coûts de transfert sur le réseau et, ainsi, augmenter la performance des requêtes sur ces données.

- Fiabilité

En dupliquant certaines données importantes sur plusieurs sites, on minimise l'impact d'une panne sur un site. De même, en cas de panne, on peut rediriger le traitement d'une requête vers un autre site disponible.

- Extensibilité

Si les besoins en espace de stockage et en puissance de traitement augmentent on peut facilement rajouter un nouveau nœud (site), sans avoir à remplacer le serveur (ex: approche Google).

## 5.b Le principe de la stratégie d'optimisation par semi-jointure

La stratégie par semi-jointure permet de réduire le coût d'une jointure en limitant la quantité de données transférées sur le réseau.

Supposons que l'on veuille calculer  $T1 \bowtie T2$  où la table  $T_i$  est située sur le site  $i$ . Au lieu de transférer une table complète d'un site à un autre, on envoie seulement les colonnes nécessaires à la jointure (la clé). Par exemple, on envoie  $\pi_{\text{clé}}(T2)$  au site 1 et on fait la jointure avec  $T1$ :

$R = T1 \bowtie \pi_{\text{clé}}(T2)$

Ceci correspond à faire la semi-jointure entre  $T1$  et  $T2$ . Ensuite, on envoie le résultat  $R$  au site 2 pour faire la jointure avec  $T2$ :

$T = R \bowtie T2 = T1 \bowtie T2$

Les données transférées sont celles de  $\pi_{\text{clé}}(T2)$  et de  $R$ , et ont une taille potentiellement moins grande que celle de  $T1$  ou de  $T2$ .

## 6. Précaution pour sauver et/ou résister aux données lors d'une catastrophe

À mon avis, la meilleure façon de sauvegarder et de restaurer des données et d'éviter l'utilisation d'une architecture centralisée en adoptant une base de données distribuée et de dupliquer les données sensibles sur plusieurs bases de données des sites différents. De ce fait, le traitement d'une requête peut être redirigé vers un autre site disponible lors d'une catastrophe.

# BDD AVANCEE - 2021(2)

## 1. Definition d une DF | utilitE de cette notion dans une BDD

### Definition

Mathématiquement, on dit que X détermine Y, ou que Y dépend fonctionnellement de X, si c'est seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y, on note  $X \rightarrow Y$ . Autrement dit, la connaissance d'une valeur de X permet de déterminer la valeur de Y pour tout tuple, X est donc la source du DF et Y est sa cible. Par conséquent, il est défini à partir des règles réagissant aux informations du système à modéliser et non à partir des exemples.

### UtilitE

La notion de DF a été introduite dès le début du relationnel par Codd afin de caractériser des relations décomposables sans perte d'information.

## 2. L'utilite de la normalisation d'une base de donnees

La normalisation est un processus de décomposition d'une table universelle en plusieurs tables en évitant le problème d'incohérence et de redondance sans perdre d'information tout en préservant les DFs et qui permet de vérifier la robustesse de leur conception pour améliorer la modélisation

### 3.a Les trois premières formes normales

1FN, 2FN, 3FN

### 3.b Relations entre ces formes normales

- **1FN**

Une relation est 1FN si et seulement si tous ses attributs ont des valeurs atomiques (non multiples). => Par définition d'une relation, toute relation est donc en 1FN

- **2FN**

Une relation est en 2FN si et seulement si:

1. La relation est en 1FN
2. Tout attribut non clé est pleinement dépendant de l'intégralité de la clé ou bien toute DF entre la clé et un attribut non clé est élémentaire. Autrement dit tout attribut non clé ne dépend pas d'une partie de la clé. Donc décomposition possible en plusieurs relations ( $R \rightarrow R_1, R_2, \dots, R_n$ ). La 2FN élimine donc les anomalies résultant des dépendances entre partie de clé et partie non clé.

=> Toute relation dont la seule clé est un attribut est une relation en 2FN

- **3FN**

Une relation est en 3FN si et seulement si:

1. La relation est en 2FN
2. Toutes les DF entre attribut clé et attribut non clé sont directes, c'est à dire pas de DF transitive. Autrement dit tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé.

## 3. Exemple & contre exemple pour chaque forme normale

### 4.a Pour que (Nom, Prenom ) soit en 1FN

Pour que (Nom, Prenom ) soit en 1FN, on le casse en deux attributs, Nom et Prénom.

### 4.b Verification en 2FN de la relation Fournisseur (nom, code-article, adresse, prix)

La relation Fournisseur (nom, code-article, adresse, prix) n'est pas en 2FN puisque 'adresse' ne dépend que d'une partie de la clé, à savoir 'nom'. On casse donc la relation en deux :

```
Fournisseur(nom-fourni, adresse)
Produit(nom-fourni, code-article, prix).
```

=> Ces relations sont maintenant en 2FN

### 4.c Verification en 3FN de la relation Voiture(num\_immat, marque, type, puissance, couleur)

La relation Voiture(num\_immat, marque, type, puissance, couleur) n'est pas en 3NF puisque 'type' attribut non clé permet de déterminer 'puissance'. On casse donc la relation en deux:

```
Voiture(num_immat, marque, type, couleur)
Type(type, puissance)
```

=> Ces relations sont maintenant en 3FN

## 5. Differences entre BD repartie et BD parallele

- **BD parallèles**

Les données peuvent être distribuées sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

- **BD réparties**

Les données sont distribuées et/ou dupliquées sur différents sites du réseau (ex: internet) qui possèdent un certain degré d'autonomie. Chaque site peut comporter une BD parallèle.

## 5. Les 2 Avantages d'une BD repartie par rapport à une architecture centralisée

- **Performance**

En rapprochant les données des applications utilisant ces données (ex : stockant les comptes des clients montréalais dans un site localisé à Montréal), on peut réduire les coûts de transfert sur le réseau et, ainsi, augmenter la performance des requêtes sur ces données.

- **Fiabilité**

En dupliquant certaines données importantes sur plusieurs sites, on minimise l'impact d'une panne sur un site. De même, en cas de panne, on peut rediriger le traitement d'une requête vers un autre site disponible.

## 6. Le principe de la stratégie d'optimisation par semi-jointure

La stratégie par semi-jointure permet de réduire le coût d'une jointure en limitant la quantité de données transférées sur le réseau. Supposons que l'on veuille calculer  $T_1 T_2$  où la table  $T_i$  est située sur le site  $i$ . Au lieu de transférer une table complète d'un site à un autre, on envoie seulement les colonnes nécessaires à la jointure (la clé). Par exemple, on envoie  $n$  clé( $T_2$ ) au site 1 et on fait la jointure avec  $T_1$ :  $R = T_1 n$  clé ( $T_2$ ) Ceci correspond à faire la semi-jointure entre  $T_1$  et  $T_2$ .

Ensuite, on envoie le résultat  $R$  au site 2 pour faire la jointure avec  $T_2$ :  $T = R T_2 = T_1 T_2$  Les données transférées sont celles de  $n$  clé ( $T_2$ ) et de  $R$ , et ont une taille potentiellement moins grande que celle de  $T_1$  ou de  $T_2$

### 7.a Definition contrainte d'intégrité

Quel que soit le modèle de données (entité association, relationnel ou autre), il existe toujours des règles du monde réel qui ne peuvent pas être exprimées par les concepts du modèle. Certaines de ces règles restreignent les valeurs que peuvent prendre les données de la base. Elles sont appelées Contraintes d'intégrité.

### 7.b Différents types de contraintes d'intégrité

- **Contraintes de domaine**

Qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table

- **Contraintes d'intégrité d'entité**

Qui précisent qu'une table doit toujours avoir une clé primaire

- **Contraintes d'intégrité référentielle**

Qui précisent les conditions dans lesquelles peuvent être ajoutées ou supprimées des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères

- **Contraintes d'intégrité quelconque**

Qui permet de spécifier que lors de toute insertion (ou suppression ou modification) d'un tuple dans telle relation telle condition doit être satisfaite sinon telle action doit être entreprise automatiquement par le SGBD, comme par exemple refuser l'insertion ou envoyer un message d'alerte.

### 7.c Exemples pour chaque type de contrainte d'intégrité

- **Contraintes de domaine**

Domaine de valeurs particulier d'un attribut: clause CHECK

- **Contraintes d'intégrité d'entité**

Identifiant: clauses PRIMARY KEY et UNIQUE Attribut obligatoire: clause NOT NULL

- **Contraintes d'intégrité référentielle**

Clause FOREIGN KEY

- **Contrainte d'intégrité quelconque**

Clause TRIGGER

# BDD AVANCEE - 2022(1)

## RSQL

```
SELECT DISTINCT nom
FROM produit, achat, client
WHERE achat.NCLI = client.NCLI AND achat.NP = produit.NP AND produit.LIB = "Vis" AND achat.QTEA >= 10000
```

```
client(#NCLI, nom, ...)
achat($NCLI, $NP, QTEA, ...)
produit(#NP, LIB, ...)
```

### 1.1 EAR correspondant

```
R1 = JOIN(achat, client, achat.NCLI = client.NCLI)
R2 = JOIN(R1, produit, R1.NP = produit.NP)
R3 = RESTRICT(R2, R2.LIB = 'Vis')
R4 = RESTRICT(R3, R3.QTEA >= 10000)
R = PROJECT(R4, R4.nom)
```

## Optimisation

```
R1 = RESTRICT(produit, produit.LIB = 'Vis')
R2 = RESTRICT(achat, achat.QTEA >= 10000)
R3 = JOIN(R1, R2, R1.NP = R2.NP)
R4 = JOIN(client, R3, client.NCLI = R3.NCLI)
R = PROJECT(R4, R4.nom)
```

### 1.2 RU correspondant

Lister le nom des clients ayant acheté plus de 10000 vis

### 1.3 M.q cette requête est loin d'être optimisée car elle nécessite un temps de calcul et un espace mémoire considérable

### 1.4 M.q on peut accélérer cette requête en utilisant des requêtes imbriquées

**Note:** Dans le langage SQL une sous-requête (aussi appelé "requête imbriquée" ou "requête en cascade") consiste à exécuter une requête à l'intérieur d'une autre requête. Une requête imbriquée est souvent utilisée au sein d'une clause WHERE ou de HAVING pour remplacer une ou plusieurs constantes.

- Optimisation en utilisant des requêtes imbriquées

```
SELECT DISTINCT nom
FROM client
WHERE client.NCLI IN (
    SELECT achat.NCLI
    FROM achat
    WHERE achat.NP = (
        SELECT produit.NP
        FROM produit
        WHERE produit.LIB = "Vis"
    ) AND achat.QTEA >= 10000
)
```

### 2.1 Les 3 premières formes normales

1FN, 2FN et 3FN

### 2.2 Relations entre ces formes

- **1FN**

Une relation est 1FN si et seulement si tous ses attributs ont des valeurs atomiques (non multiples). => Par définition d'une relation, toute relation est donc en 1FN

- **2FN**

Une relation est en 2FN si et seulement si:

1. La relation est en 1FN
2. Tout attribut non clé est pleinement dépendant de l'intégralité de la clé ou bien toute DF entre la clé et un attribut non clé est élémentaire. Autrement dit tout attribut non clé ne dépend pas d'une partie de la clé. Donc décomposition possible en plusieurs relations ( $R \rightarrow R_1, R_2, \dots, R_n$ ). La 2FN élimine donc les anomalies résultant des dépendances entre partie de clé et partie non clé.

=> Toute relation dont la seule clé est un attribut est une relation en 2FN

- **3FN**

Une relation est en 3FN si et seulement si:

1. La relation est en 2FN
2. Toutes les DF entre attribut clé et attribut non clé sont directes, c'est à dire pas de DF transitive. Autrement dit tout attribut non clé ne dépend pas d'un autre attribut non clé mais directement de la clé.

### 3. Définition d'une DF | utilité de cette notion dans une BDD

#### Definition

Mathématiquement, on dit que X détermine Y, ou que Y dépend fonctionnellement de X, si c'est seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y, on note  $X \rightarrow Y$ . Autrement dit, la connaissance d'une valeur de X permet de déterminer la valeur de Y pour tout tuple, X est donc la source du DF et Y est sa cible. Par conséquent, il est défini à partir des règles réagissant aux informations du système à modéliser et non à partir des exemples.

#### Utilité

La notion de DF a été introduite dès le début du relationnel par Codd afin de caractériser des relations décomposables sans perte d'information.

### 4.1 Définition contrainte d'intégrité

Quel que soit le modèle de données (entité association, relationnel ou autre), il existe toujours des règles du monde réel qui ne peuvent pas être exprimées par les concepts du modèle. Certaines de ces règles restreignent les valeurs que peuvent prendre les données de la base. Elles sont appelées Contraintes d'intégrité.

### 4.2 Différents types de contrainte d'intégrité

- **Contraintes de domaine**

Qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table

- **Contraintes d'intégrité d'entité**

Qui précisent qu'une table doit toujours avoir une clé primaire

- **Contraintes d'intégrité référentielle**

Qui précisent les conditions dans lesquelles peuvent être ajoutées ou supprimées des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères

- **Contraintes d'intégrité quelconques**

Qui permet de spécifier que lors de toute insertion (ou suppression ou modification) d'un tuple dans telle relation telle condition doit être satisfaite sinon telle action doit être entreprise automatiquement par le SGBD, comme par exemple refuser l'insertion ou envoyer un message d'alerte.