Conception et exploitation d'une base de données

Equipe Bases de Données, Université LYON 1, LIRIS.

LYON 1 - Informatique - Laboratoire LIRIS

26 octobre 2025

Plan du cours

Contexte

- Nous sommes dans l'ère du BIG DATA
  - Les données sont générées, récoltées, exportées, échangées, vendues
  - Puis mises en entrée d'algorithmes d'analyse ou de modélisation
    - Requêtes, Statistiques, Data Mining, Apprentissage...
  - Enjeux commerciaux, scientifiques, sécuritaires considérables - et moyens qui vont avec.
  - Les données sont un bien précieux de chaque organisation
    - Pour le fonctionnement de quasiment toutes les "applications"
    - Pour améliorer l'efficacité et la qualité de l'entreprise
    - Pour guider les évolutions
    - Pour satisfaire à des exigences légales

Fortes exigeances sur les données

- Stocker les données de façon fiable
- Gérer des accès partagés
- Pouvoir effectuer des recherches complexes et rapides
- être évolutif, réactif, sécurisé
- et bien d'autres choses !

Il faut des outils professionnels : les SGBD

- Exemples : PostgreSQL, Oracle, SQL Server, ...

Définitions

Définition

Une Base de Données (BD) est une collection de données électroniques stockées de façon pérenne et reliées entre elles par un contexte applicatif.

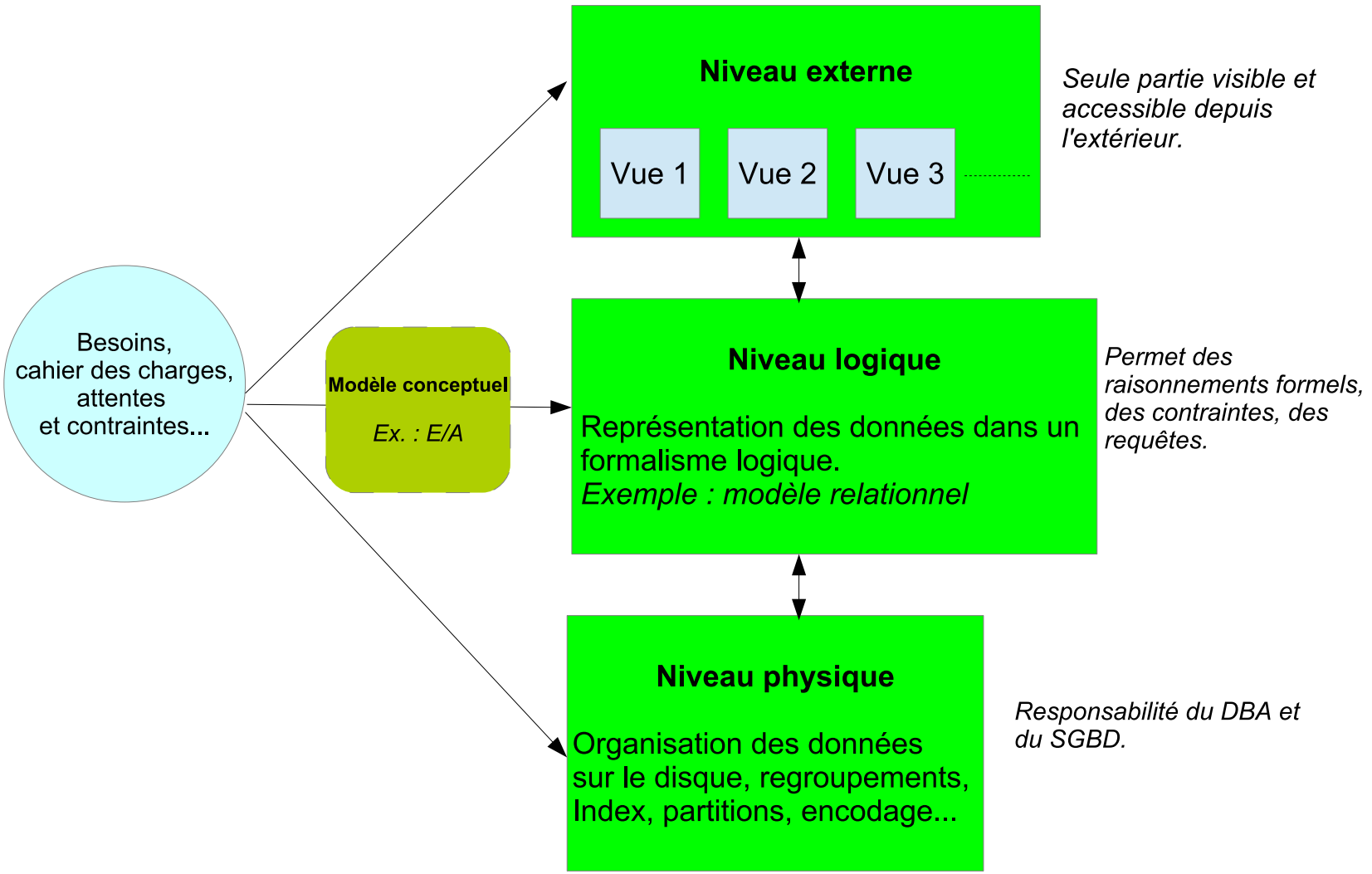
- Quelques exemples :
  - BD de la scolarité de l'université
  - BD des restaurants lyonnais pour la gestion d'un guide
  - BD des ventes d'une entreprise

Contenu d'une BD

Une BD est toujours justifiée par des besoins !

- Ce sont les besoins applicatifs qui guident le contenu, l'organisation, la portée d'une BD.
- La conception des BD est donc un sous-problème de la conception logicielle...
- ... mais plusieurs logiciels pourront accéder à une même base de données !

Niveaux (souhaitables) d'abstraction et d'interactions



Indépendance entre les niveaux

Indépendance logique

- On peut modifier le modèle logique en conservant les mêmes vues externes.
  - Séparation entre BD et applications
  - Permet des évolutions "indépendantes" des deux niveaux

Indépendance physique

- On peut modifier des choix physiques en conservant le modèle logique
  - Sémantique conservée, requêtes inchangées
  - Tout en opérant des réglages physiques

Indépendance assez aboutie dans les SGBD relationnels.

Conception d'une base de données

Conception = construire une base de données à partir d'un cahier des charges.

Etape cruciale à considérer avec soins !

- Conception = établir le modèle logique
- De façon à modéliser au mieux les données réelles
- Et pouvoir répondre aux besoins (requêtes)
- Interactions nécessaires avec les experts des données et les concepteurs d'applications
  - Non spécialistes des bases de données
  - Le modèle conceptuel facilitera ces échanges

Une bonne conception est certes coûteuse, mais nécessaire !

Pour les applications

- Avoir des résultats justes, complets, performants aux requêtes.

Pour la gestion des données

- Réduire le travail des administrateurs, les sources d'incohérence
- Faciliter la maintenance et les évolutions

Pour la science des données

- L'analyste doit comprendre la sémantique des données, les interactions, l'interprétation des valeurs manquantes...
- Une BD bien conçue est bien plus facile à analyser

Plan du cours

Un formalisme graphique de travail

- E/R (Entity-Relationship) en anglais.
- Langage graphique pour élaborer un Modèle Conceptuel de Données (MCD)
- Modélise les entités, leurs attributs et leurs associations (interactions)
- Très intuitif, simplifie les échanges autour du cahier des charges
- Bien intégré à la démarche de conception logicielle, proximité avec un diagramme ce classe UML.
- Fourni une documentation précieuse pour l'évolutivité
- Se traduit automatiquement ver le modèle logique (relationnel)

Un formalisme graphique de travail

Attention à ne pas tomber dans la facilité

- Simple langage sans mécanisme de raisonnement
- Des choix dépendent du concepteur, de son expérience
- Pouvoir d'expression limité -> tendance à trop simplifier

Rester proche des besoins

- Démarche itérative
- Bien critiquer ses choix et lister les manques pour la suite
- Importance de connaître les bonnes pratiques

Les élements du langage

- Entités
- Associations
  - Binaires (entre deux entités) ou bien n-aires.
- Attributs
  - Décrivent les entités, où les associations
  - Certains attributs d'entité sont des Identifiants.
- Deux types particuliers d'entités [1], presque indispensables :
  - Entité faibles
  - Entités spécialisées

1. appartiennent au modèle E/A dit "étendu"

Entités et Classes d'entités

- Entité :
  - Une "chose" du monde réel qu'on cherche à modéliser
  - Exemple
    - Un étudiant, un diplôme...
- Classe d'entités
  - Modélisation commune d'entités
  - Exemple : "Un étudiant est représenté par son num, son nom et son prenom."
  - Une entité appartient à une classe d'entité

Par abus de langage, Entité = Classe d'entité.

Associations et Classes d'Association

- Association :
  - Une relation entre deux ou plusieurs entités.
  - Exemple :
    - Tom *est inscrit* en Master Informatique
- Classe d'Association :
  - Décrit de façon commune un ensemble d'associations
    - Exemple : Un étudiant s'inscrit dans un diplôme

Par abus de langage, Assocation = Classe d'association.

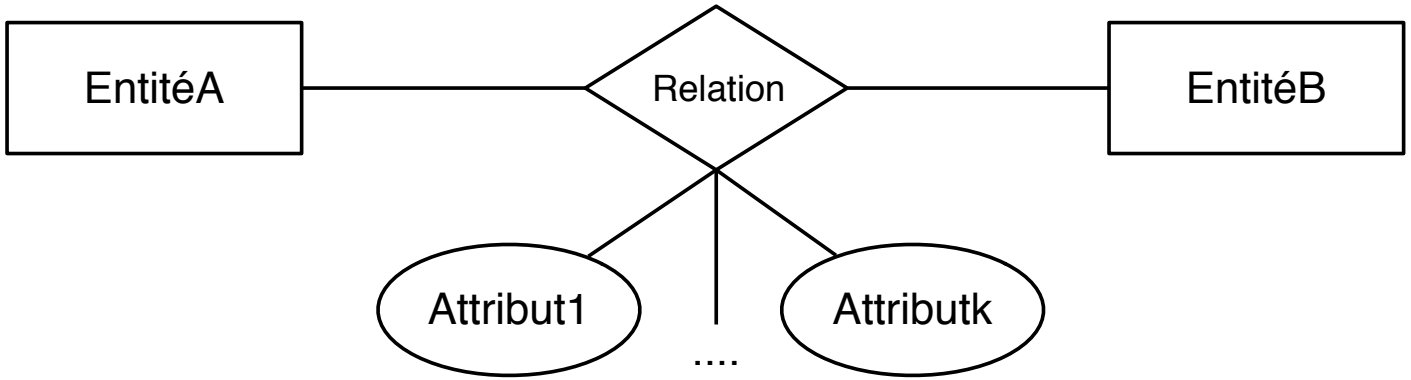
Attributs

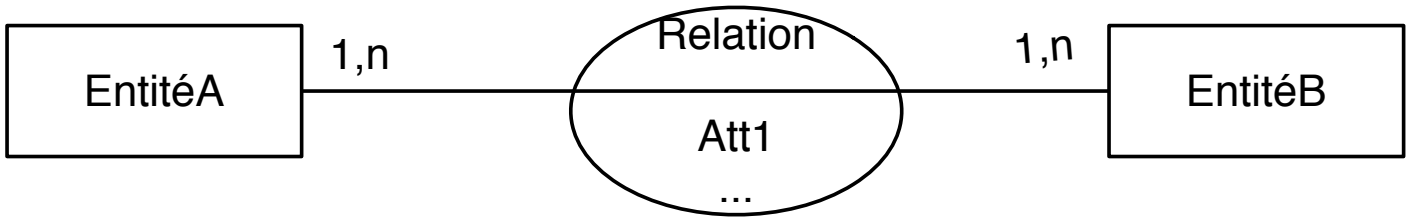
- Attribut :
  - Propriété d'une entité ou d'une association prend ses valeurs dans un domaine de valeurs de type simple (caractère, chaîne de caractères, entier, date).
  - Exemple :
    - L'entité Etudiant a pour attributs Num, Nom et Prenom
    - L'association "inscrit" peut avoir pour attribut la date d'inscription
  - On peut accepter les attributs multivalués, marqués du symbole '*'. Ils peuvent alors prendre une liste de valeurs.
    - Exemple : un document peut avoir plusieurs mots clés, une personne plusieurs prénoms
    - Attention : chaque valeur reste bien atomique.

Au moins un ensemble d'attributs permet d'identifier de façon unique une entité. On souligne cet identifiant [2] .

2. Un attribut multivalué ne peut pas être identifiant

Formalisme de représentation des associations



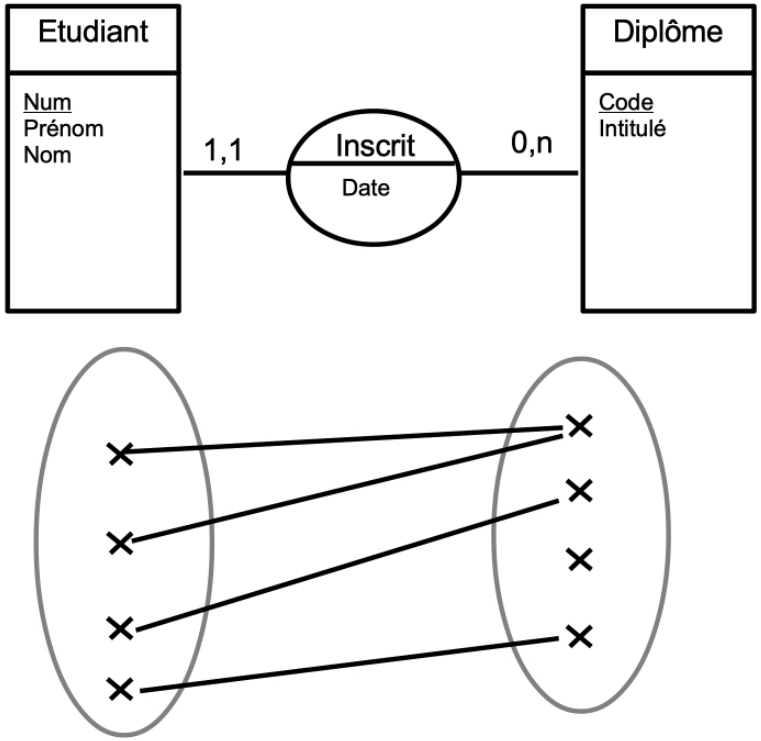


Connectivité et participation aux associations

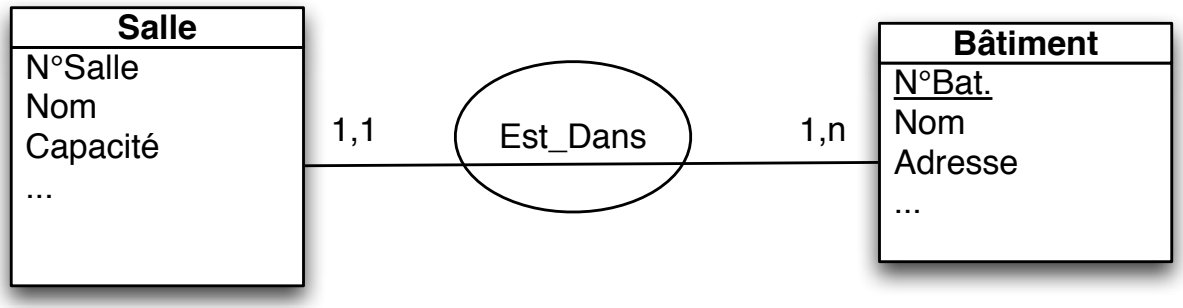
Dans une association entre E_1 et E_2, on doit définir :

- A combien d'entité E_2 peut se connecter une entité E_1 et inversement : c'est la connectivité de chaque classe d'entité. Elle vaut soit 1, soit N
- Si chaque entité E_1 (ou E_2) est obligée de participer à l'association : c'est la participation de chaque classe d'entité. Elle vaut soit 0, soit 1.
- Exemples :
  - Un étudiant DOIT être inscrit dans au moins une formation (participation obligatoire)
  - Un étudiant doit être inscrit AU PLUS dans une formation (connectivité simple)
  - Une formation peut avoir PLUSIEURS étudiants (connectivité multiple)
  - Une formation peut n'avoir aucun étudiant (participation optionnelle)

Connectivité et participation aux associations



Entités Faibles



Que se passe-t-il s'il y a deux salles 1 dans deux bâtiments différents ?

- L'attribut "N Salle" ne permet pas d'identifier une salle.
- Il faut savoir de quel bâtiment il s'agit.
- Lien existentiel : la salle n'existe que si le bâtiment existe.

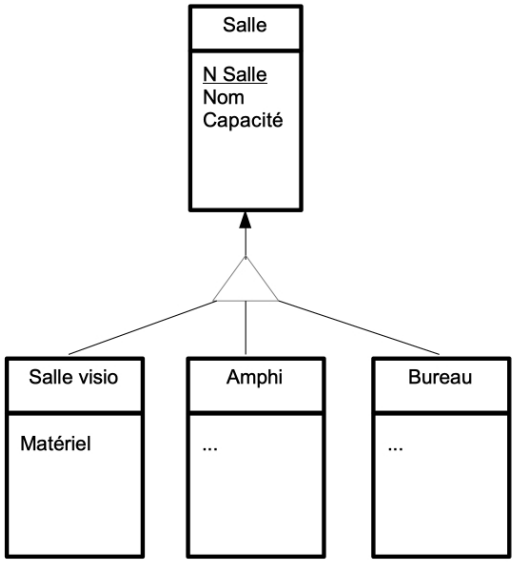
Entités Faibles

Pour savoir de quelle salle on parle, il faut connaître dans quel bâtiment elle est. Une salle est donc "identifiée" par : un numéro de salle $NumSalle$ (identifiant local) et un numéro de bâtiment $NumBat$.

Spécialisation / Généralisation

- Une entité représente un cas particulier d'une autre entité
- Les entités "filles" héritent des attributs des entités "parents", y compris de l'identifiant.
- Donc il ne faut répéter aucun de ces attributs ni identifiants.
- Exemple :
  - Une salle de visio, en plus des autres, possède un matériel spécifique, des identifiants de connexion, un modes de réservation, etc. . .
  - Il faut donc les modéliser de façon distincte des autres salles
  - Sans perdre de vue que c'est bien une salle !

Exemple



Attention : contrairement à une entité faible qui possède un identifiant local souligné, une entité spécialisée n'a JAMAIS d'attribut souligné. Son identifiant provient totalement de l'entité "mère" à laquelle elle est reliée.

Associations n-aires

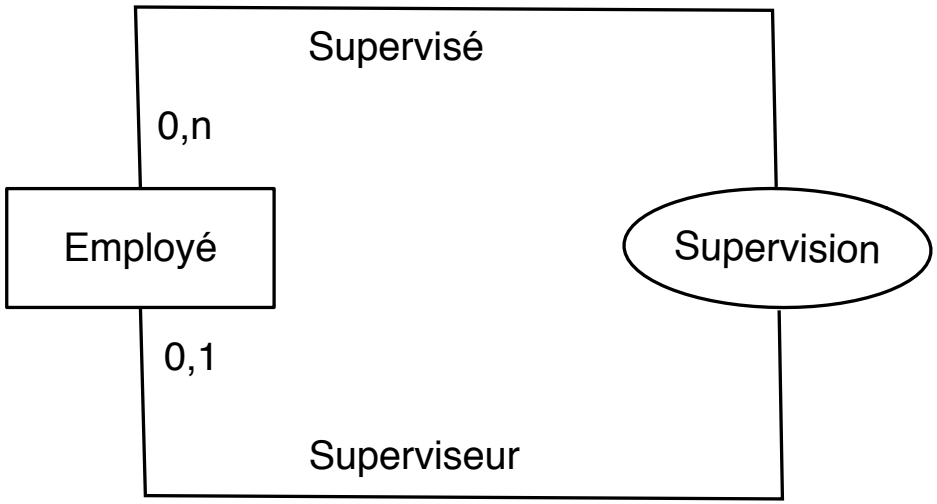
Il est possible de faire interagir trois entités ou plus dans une même association :

- C'est parfois une première intention naturelle
- Mais beaucoup de "fausses" associations n-aires, faites un peu rapidement
- Tous les sous-ensembles d'entités peuvent-ils se répéter ?
- Si la réponse est non, utiliser des associations d'associations (agrégations)

Une agrégation consiste à considérer les couples d'une association binaire commme des "pseudo-entités", de façon à les associer à leur tour à d'autres entités.

Associations réflexive

- Une association réflexive associe des entités de même type.
- Il faut alors ajouter *un rôle* à chaque élément de l'association, pour lever les ambiguïtés.



Pour aller plus loin dans les contraintes

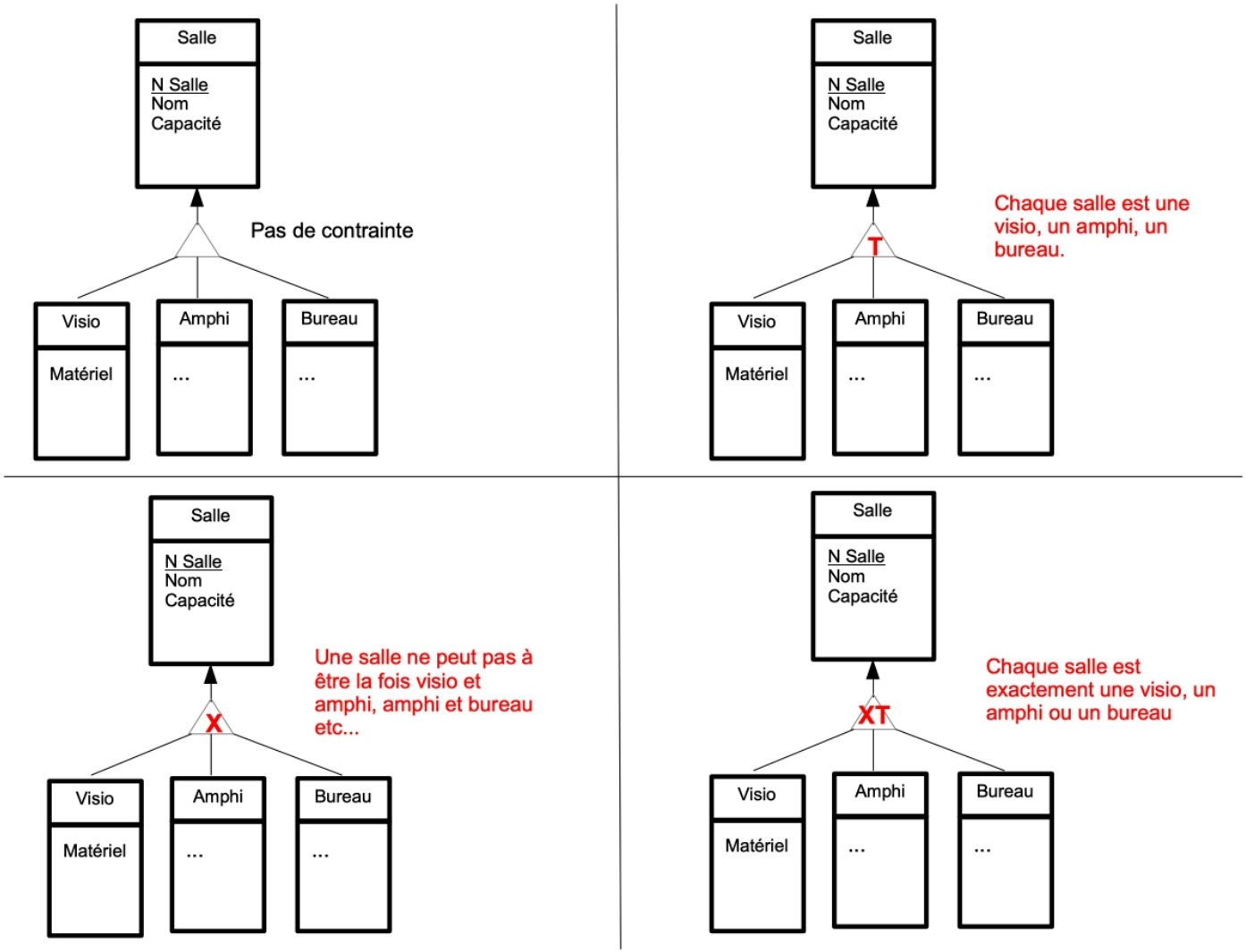
Sur les liens de spécialisations

- Contrainte de totalité (couverture) notée T : toutes les entités correspondent à au moins une spécialisation.
- Contrainte d'exclusivité notée X : une entité ne peut pas être dans deux spécialisations.
- Contrainte de partition notée XT, toutes les entités correspondent à exactement une spécialisation

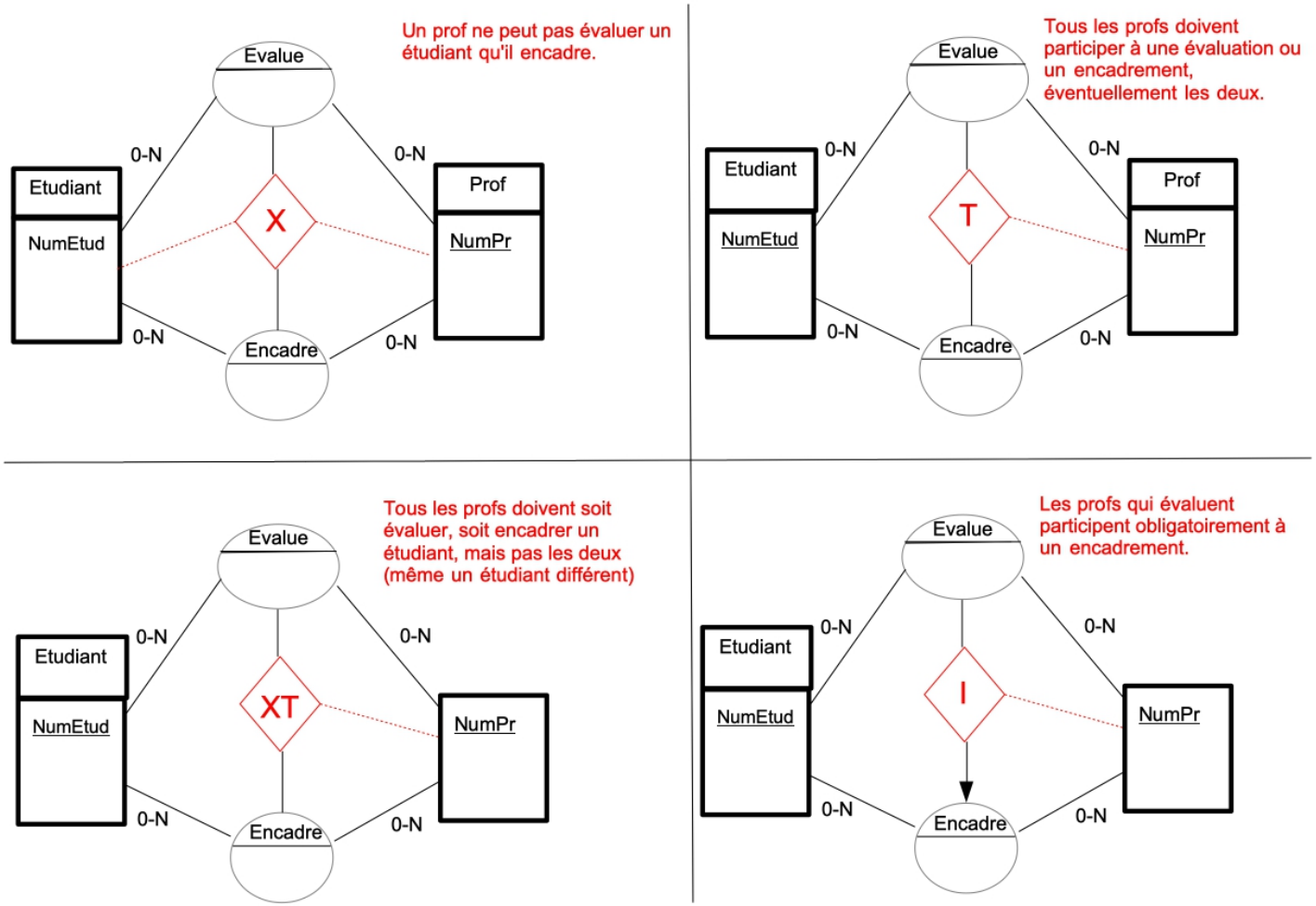
Entre des associations qui impliquent les mêmes entités

- Totalité T, exclusivité X et partition XT sur les participations
- Contrainte d'inclusion I entre les associations A_1 et A_2 : les entités participant à A_1 participent aussi à A_2.
- Contrainte d'égalité = : inclusion dans les deux sens.

Exemples de contraintes avancées



Exemples de contraintes avancées



Remarques

On ne peut pas tout Modéliser

- Beaucoup de subjectivité ! Et des limites d'expressivité.
- Lister les spécifications non modélisées
- Elles permettront d'affiner sur le modèle logique plus tard
- Et enventuellement modifier le schéma E/A en conséquence

Quelques points de vigilance

- Le graphe final doit être connexe. Il manque probablement quelque chose sinon.
- Une association n'a jamais d'attribut souligné. Si vous avez "besoin" de le faire, c'est qu'il vous manque une entité...
- Un cycle dans le graphe n'est pas une erreur, mais mérite de l'attention.
- Idem pour les associations

Exercices

Une entreprise souhaite modéliser les informations suivantes : Chaque employé est identifié par un numéro de sécurité sociale unique. Chaque employé a un nom, un salaire et un numéro de téléphone et travaille dans un département, identifié par son nom. On stocke les enfants des employés pour les occasions sociales ; chaque enfant est identifié par un nom et une date de naissance. Un enfant est associé à un ou plusieurs employés.

Quelques exercices

L'université gère sa scolarité, avec un ensemble de formations et de modules qui sont proposés. Il y a deux types d'étudiants, qui ont tous un numéro unique, un nom et une adresse : ceux en formation initiale s'inscrivent à une formation unique chaque année. Ils suivront alors tous les modules qui composent cette formation. Mais les étudiants en formation continue
chaque étudiant (qui a un numéro d'étudiant unique, un nom et une adresse) en formation initiale possède une inscription unique, chaque année, dans l'une des formations de l'université. Les formations sont constituées de modules, qu'elles peuvent d'ailleurs partager. Il existe aussi des étudiants inscrits sous le statut de formation continue : ces étudiants ne s'inscrivent pas dans une formation mais dans un ensemble de modules. Pour chaque module on stocke le prix qui est payé par cet étudiant.

Quelques exercices

Une entreprise en ligne commercialise et livre des produits. Chaque produit appartient à une catégorie ; les catégorie sont organisées de façon hiérarchique (sous-catégories). Les commandes sont faites par les clients et enregistrées dans la base au moment de la validation du panier ; un client ne peut donc avoir deux commandes au même instant. Chaque commande possède une adresse de livraison. Une commande correspond à un ensemble de produits, on repère le prix final auquel ce produit est cédé dans cette commande.
Chaque commande fait l'objet de livraisons ; il peut y avoir plusieurs livraisons pour une commande - mais pas à la même date. On notera pour chaque livraison le nom du transporteur.

Quelques exercices

1) Un guide de restaurant souhaite se donner d'une base de données. Chaque restaurant est identifié par un numéro unique. Chaque restaurant a un nom, une adresse et un type de cuisine (italienne, japonaise, etc.). Chaque menu est identifié par un numéro unique, possède un nom et une description. Un menu est proposé par un seul restaurant, mais un restaurant peut proposer plusieurs menus. Un menu est composé de plats qui ont un nom unique et une description. Un plat peut appartenir à plusieurs menus.
2) On gère maintenant les réservations. Via une application, des clients à qui on attribue un numéro unique peuvent réserver, à une date et heure donnée, dans un (un seul) des restaurants. Le client choisit les menus (il peut venir accompagné) au moment de la réservation.

Plan du cours

Plan du chapitre

Qu'est ce qu'un modèle de données ?

Un formalisme pour représenter et interroger les données

- Une structure pour représenter les données
- Des contraintes pour garantir la sémantique du cahier des charges
- Des langages pour interroger et modifier les données

Les principaux modèles de données

- *Modèle relationnel*
  - *Structure* : des relations entre des tuples et des attributs.
  - *Contraintes* : clés (identifiants de tuples, clés étrangères (références à des tuples), contraintes de domaines.
  - *Langages* : algèbre relationnelle, calcul relationnel, SQL, clauses de Horn sans récursion.
- *Modèle déductif*
  - *Structure* : celle du modèle relationnel à laquelle on ajoute des règles de déduction.
  - *Contraintes* : les mêmes que le modèle relationnel
  - *Manipulation* : langages logiques comme *Datalog*. Contrairement aux langages du modèle relationnel, il admet la récursivité.

Les principaux modèles de données

- *Modèle de graphe (e.g., RDF)*
  - *Structure* : graphe orienté et étiqueté
  - *Contraintes* : un identifiant pour chaque nœud, un mécanisme de référence entre des nœuds
  - *Manipulation* : parcours de graphes, SPARQL.
- *Modèle hiérarchique (e.g., XML)*
  - *Structure* : arborescente (forêt d'arbre)
  - *Contraintes* : un identifiant pour chaque nœud, un mécanisme de référence entre des nœuds
  - *Manipulation* : navigation hiérarchique, XPATH, XQUERY.

- *Modèle objet*
  - *Structure* : logique objet, soit des classes, des objets, des attributs et des méthodes. Peut être vu comme un graphe orienté.
  - *Contraintes* : identifiant pour les objets, référence entre objets.
  - *Manipulation* : extensions de SQL comme OSQL ou OQL.
- *Modèle Entité/Association*
  - *Structure* : Entités (avec des attributs) et associations entre des entités.
  - *Contraintes* : identifiants d'entités, cardinalités sur les associations, contraintes avancées
  - *Manipulation* : aucun (c'est un langage de modélisation).

Les bases où sont créées et modifiées les données sont très majoritairement relationnelles depuis les années 1980.

Notations

- L'ensemble $\{A; B; C\}$ sera noté ABC
- On n'écrira jamais BCA par convention : on utilise l'ordre alphabétique.
- Un ensemble n'a NI ordre, NI répétitions
- Soit les ensemble $X = ABC$ et $Y = BD$, alors leur union $X \cup Y$ sera noté $XY = ABCD$.

Le modèle relationnel : Intuition

Étudiants	*NUMETUD*	*NOMETUD*	*PRENOMETUD*	*AGE*	*FORMATION*
	28	Codd	Edgar	20	3
	32	Armstrong	William	20	4
	53	Fagin	Ronald	19	3
	107	Bunneman	Peter	18	3

Enseignants	*NUMENS*	*NOMENS*	*PRENOMENS*	*GRADE*
	5050	Tarjan	Robert	PR
	2123	Mannila	Heikki	MCF
	3434	Papadimitriou	Spiros	PR
	1470	Bagan	Guillaume	CR

Encadre	*NUMENS*	*NUMETUD*	*DATE*
	5050	53	2005
	3434	28	2020
	5050	28	2015
	2123	32	2019

TABLE – Exemple de base de données relationnelle

Plan du chapitre

Le modèle relationnel : structure

- Soit $\mathcal{U}$, un ensemble infini dénombrable de *noms d'attributs* ou simplement *attributs*, appelé univers.
- Soit $\mathcal{D}$ un ensemble infini dénombrable de constantes (ou valeurs).
- Soit $A \in \mathcal{U}$ un *attribut*, le domaine de A est un sous-ensemble de $\mathcal{D}$, noté $DOM(A)$.

Schémas de relations et de bases de données

- Un schéma de relation R est un ensemble fini d'attributs (donc $R \subseteq \mathcal{U}$).
- Un schéma de base de données $\mathbf{R}$ est un ensemble fini de schémas de relation.

Le modèle relationnel : structure

Tuple, Relation et Base de Données

- Soit $R = A_1 ... A_n$ un schéma de relation. Un tuple sur R est un élément du produit cartésien $DOM(A_1) \times \ldots \times DOM(A_n)$.
- Une relation r sur R (appelée aussi instance ou vulgairement table) est un ensemble *fini* de tuples.
- Une base de données **d** sur un schéma de base de données $\mathbf{R} = \{R_1, ..., R_n\}$ est un ensemble fini de relations $\{r_1, ..., r_n\}$ définies sur les schéma de relation de $\mathbf{R}$.

- Si t est un tuple défini sur un schéma de relation R, et X un sous-ensemble de R, on peut restreindre t à X en utilisant la projection, notée $t[X]$ qui est la restriction de t à X.

synthèse

Schéma de la relation « etudiants » : ensemble de 5 attributs

Relation Etudiants : ensemble de 4 tuples

Un tuple de la relation « etudiants »

Étudiants	NUMETUD	NOMETUD	PRENOMETUD	AGE	FORMATION
	28	Codd	Edgar	20	3
	32	Armstrong	William	20	4
	53	Fagin	Ronald	19	3
	107	Bunneman	Peter	18	3

Enseignants	NUMENS	NOMENS	PRENOMENS	GRADE
	5050	Tarjan	Robert	PR
	2123	Mannila	Heikki	MCF
	3434	Papadimitriou	Spiros	PR
	1470	Bagan	Guillaume	CR

Encadre	NUMENS	NUMETUD	DATE
	5050	53	2005
	3434	28	2020
	5050	28	2015
	2123	32	2019

Modèle relationnel : contraintes

Le modèle supporte trois types de contraintes : clés, clés étrangères et contraintes de domaines.
Une clé définie sur un schéma de relation est un ensemble d'attributs sur lequel chaque tuple prendra une valeur unique.

- Exemple : $NUMETUD$ est une clé du schéma de la relation $Etudiants$.
- Un schéma de relation a toujours au moins une clé par défaut : l'ensemble de tous les attributs du schéma, puisqu'aucune relation ne peut avoir deux fois le même tuple.
- Une clé est dite minimale si elle ne contient aucune autre clé. Sinon, on parle de superclé.

En pratique, on définit une "clé primaire" choisie parmi les clés minimales, qui sera utilisée par défaut pour identifier les tuples.

Modèle relationnel : contraintes (suites)

Une clé étrangère dans un schéma de relation est un ensemble d'attributs qui permet de faire référence à une clé d'un autre schéma, et donc, pour chaque valeur qui sera affectée, à un tuple précis d'une autre relation.

- Exemple : $NUMETUD$ est une clé étrangère de la relation $Encadre$ faisant référence à l'attribut $NUMETUD$ de $Etudiants$.

Précisément, la contrainte de clé étrangère est une contrainte d'existence : si un tuple prend une valeur sur une clé étrangère, alors cette valeur doit exister parmi les valeurs de la clé référencée.

Modèle relationnel : contraintes (fin)

Une contrainte de domaine est une restriction des valeurs possibles qu'on peut affecter aux attributs. La définition même des domaines, (par des types de données, ou des listes de valeur) de chaque attribut est une telle forme de contrainte ; mais on peut rajouter des conditions du type : pour chaque tuple, la valeur de l'attribut A doit être supérieure à la valeur de l'attribut B.
Le fait d'autoriser ou non les valeurs NULL (valeurs manquantes) pour des attributs est aussi un exemple de contrainte de domaine.

Plan du chapitre

Traduction d'un schéma EA en relationnel

Traduction des entités

Chaque entité devient une relation.

- Les identifiants de l'entité deviennent des clés
- Entités spécialisées et entités faibles
  - La clé de l'entité forte est ajoutée en clé étrangère
  - Pour les entités spécialisées, cette clé étrangère est la clé.
  - Pour les entités faibles, cette clé étrangère PLUS la clé locale constituent la clé.

Traduction d'un schéma EA en relationnel

Traduction des associations

Chaque association devient une relation.

- Les attributs de l'association sont attributs de la relation
- On ajoute comme attributs une dupplication de la clé de chaque entité participante, avec une contrainte de clé étrangère.
- La définition de la clé dépend de la connectivité
  - la clé étrangère correspondant à chaque entité qui possède une connectivité de 1 devient une clé.
  - Si toutes les connectivités sont N, alors la clé est composée de toutes les clés étrangères des entités participantes.

Solution alternative

Lorsque la connectivité d'une entité à une association est 1, une autre solution consiste à placer une clé étrangère dans cette entité qui fait référence à la clé de l'autre entité.

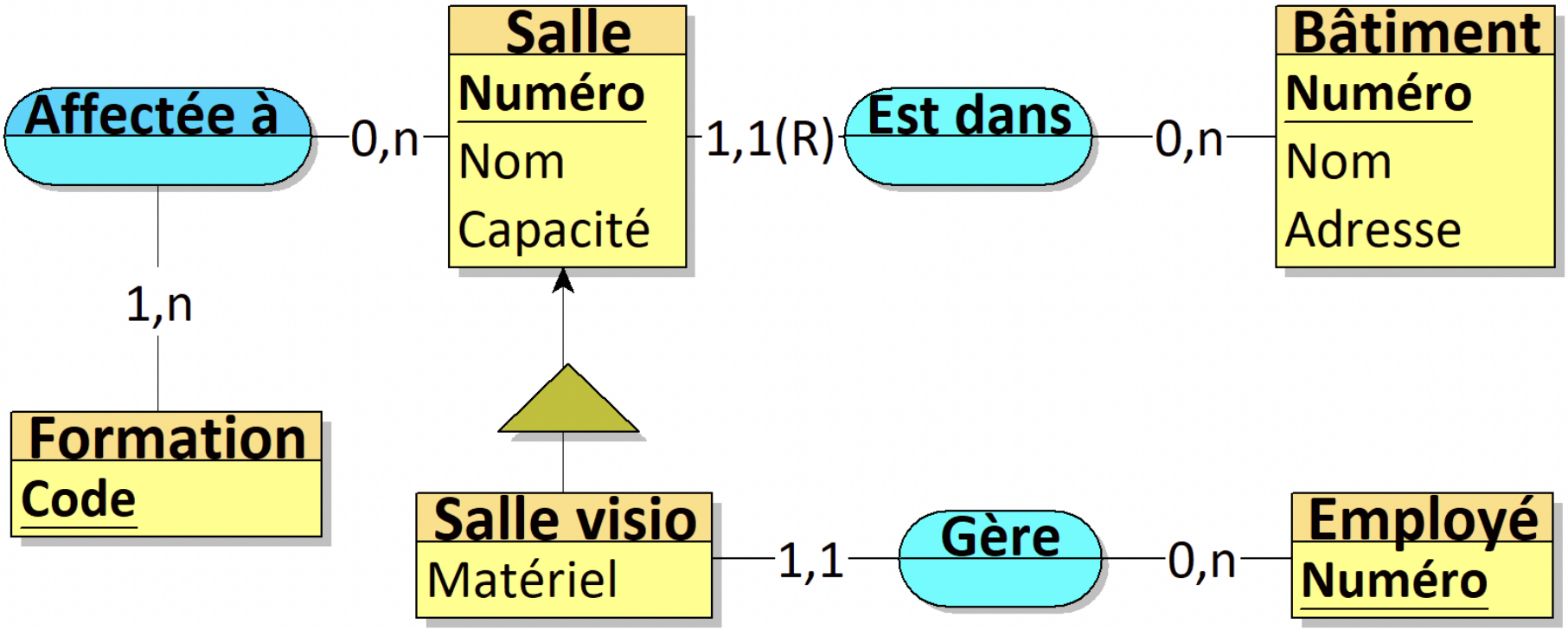
Traduction d'un schéma EA en relationnel

Le travail n'est pas fini !

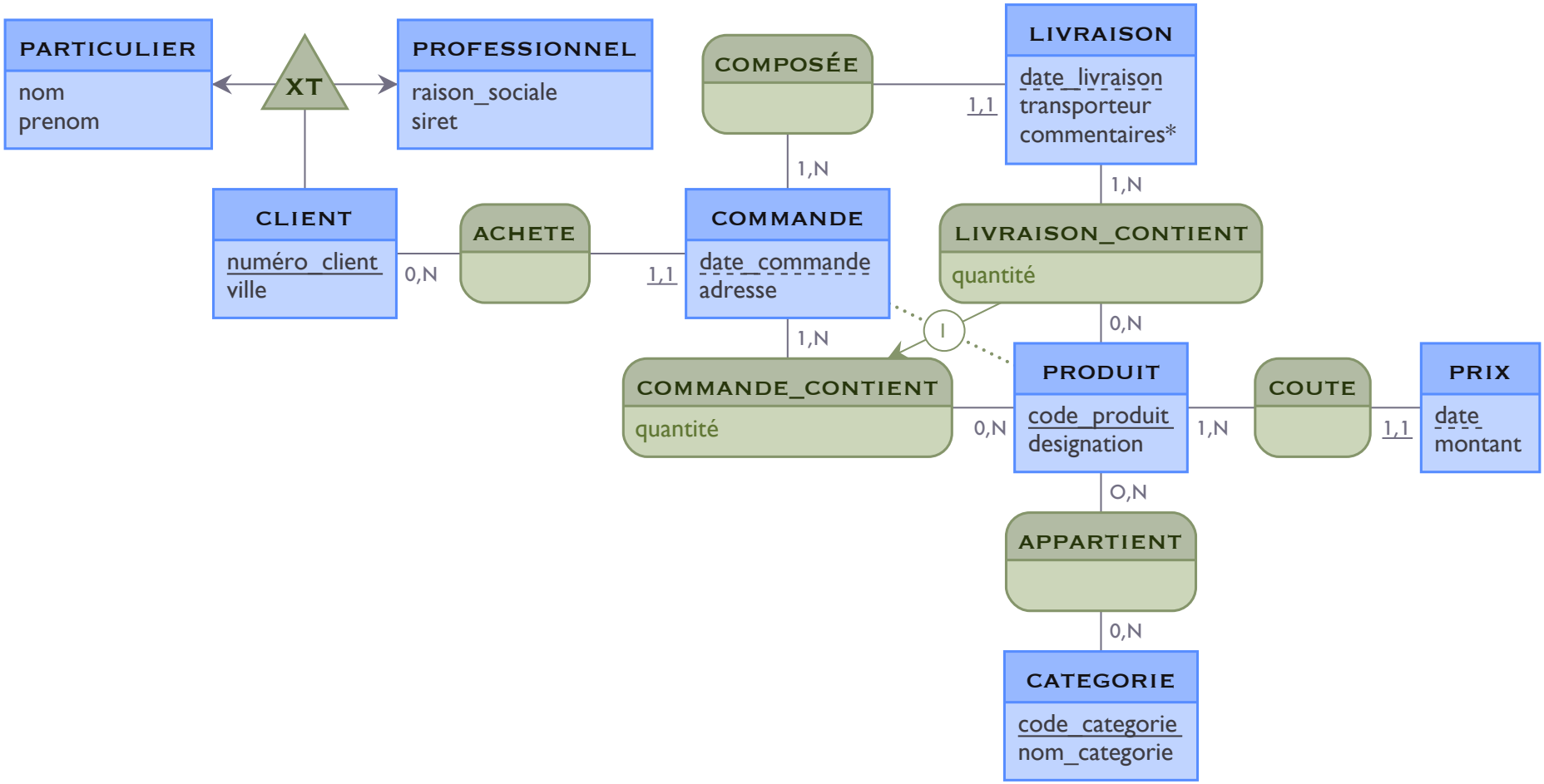
- On peut simplifier, abandonner des relations, ... En justifiant et en documentant.
- Intégrer les contraintes avancées (T,X,...) :
  - Certaines se traduiront pas des clés étrangères, mais à voir au cas par cas. . .
  - Si une contrainte ne se traduit ni en clé, ni en clé étrangère ni en contrainte de domaine, alors il faut la programmer spécifiquement sur le serveur de BD.
- Une conception plus fine peut être poursuivie en relationnel
  - Normalisation et optimisation du schéma (compétences avancées)
  - Retour éventuel pour faire évoluer le schéma E/A

Toujouts maintenir un diagramme E/A "commenté" équivalent au modèle relationnel implémenté.

Exercices (1)


Affectée à
0,n
Salle
Numéro
Nom
Capacité
1,1(R)
Est dans
0,n
Bâtiment
Numéro
Nom
Adresse
1,n
Formation
Code
Salle visio
Matériel
1,1
Gère
0,n
Employé
Numéro

Exercices (2)


PARTICULIER
nom
prenom
XT
PROFESSIONNEL
raison_sociale
siret
CLIENT
numéro_client
ville
0,N
ACHETE
1,1
COMMANDE
date_commande
adresse
COMPOSÉE
1,N
1,1
LIVRAISON
date_livraison
transporteur
commentaires*
1,N
LIVRAISON_CONTIENT
quantité
0,N
I
1,N
COMMANDE_CONTIENT
quantité
0,N
PRODUIT
code_produit
designation
1,N
COUTE
1,1
PRIX
date
montant
O,N
APPARTIENT
0,N
CATEGORIE
code_categorie
nom_categorie

Plan du chapitre

Modèle relationnel : Langages

- Langages théoriques d'interrogation des données :
  - Langage procédural : algèbre relationnelle
  - Langage déclaratif (logique) : calcul relationnel (tuple ou domaine), Datalog.
- Langage implémenté pour l'interrogation et la manipulation des données
  - Structured Query Langage (SQL)
  - SQL est l'implémentation du calcul relationnel pour la partie interrogation.
  - Mais nombreuses extensions, et possibilité d'ajouter des fonctions.

On manipule des relations

Une requête prend une plusieurs relations en entrée, et retourne une relation. On peut donc interroger des sous-requêtes ; toutefois seul Datalog est récursif.

Algèbre Relationnelle

Soient des relations r, r_1, r_2 définies respectivement sur les schémas R, R_1, R_2.

- Sélection ($\sigma_C(r)$) filtre les tuples de r selon la condition $C|D$.
  - On obtient un sous ensemble de r
- Projection ($\pi_X(r)$) ne conserve que les attributs de X ($X \subseteq R$)
  - On écarte les colonnes qui n'interessent pas la requête
- Jointure ($r_1 \bowtie r_2$) "combine" entre eux les tuples de r_1 et r_2
  - On combine les tuples qui sont égaux sur $R_1 \cap R_2$.
- Renommage ($\rho_{[X/X']}(r)$) opération de renommage des attributs.

Algèbre Relationnelle (suite)

Puisque les relations sont des ensembles de tuples, on bénéficie en plus de tous les opérateurs ensemblistes.

- A condition d'avoir $R = S$:
  - Différence ($r_1 \setminus r_2$).
  - Intersection ($r_1 \cap r_2$).
  - Union ($r_1 \cup r_2$).
- A condition d'avoir $R \cap S = \emptyset$
-
  - Produit cartésien ($r_1 \times r_2$). La relation obtenue est sur le schéma $R_1 \cup R_2$.

Pour les conditions sur le schéma, on peut les "forcer" par le renommage préalable

Exemples

Répondre aux questions suivants par une requête algébrique.

- Quel est le prenom et le nom de tous les étudiants
- Quel est le prenom et le nom des enseignants qui sont PR
- Quel est le nom des enseignants qui encadrent l'étudiant 53 ?
- Quels est le num des étudiants qui n'ont pas d'encadrant.
- Lister le prenom et nom de tous les étudiants et enseignants

Exercices

Répondre aux questions suivants par une requête algébrique.

- Quel est le prenom et le nom de tous les étudiants
  - $\pi_{PRENOMETUD,NOMETUD}(Etudiants)$
- Quel est le prenom et le nom des enseignants qui sont PR
  - $\pi_{PRENOMENS,NOMETUD}((\sigma_{GRADE='PR'}(Enseignants)))$
- Quel est le nom des enseignants qui encadrent l'étudiant 53 ?
  - $\pi_{NOMENS}(Enseignants \bowtie \sigma_{NUMETUD=107}(Encadre))$
- Quels est le num des étudiants qui n'ont pas d'encadrant.
  - $\pi_{NUMETUD}(Etudiants) \setminus \pi_{NUMETUD}(Encadre)$
- Lister le nom de tous les étudiants et enseignants
  - $\rho_{NOMETUD/NOM}(\pi_{NOMETUD}(Etudiants)) \cup$
    $\rho_{NOMENS/NOM}(\pi_{NOMENS}(Enseignants))$

Chaque requête est un arbre parenthésé, et plusieurs requêtes peuvent répondre à une même question.

Calcul Relationnel à Variable Tuples

- Syntaxe :

$$\{x^{(n)} | F(x)\}$$

où $x^{(n)}$ est un n-uplet (c'est à dire un tuple à n champs) et F est une formule logique du premier ordre ; $F(x)$ exprime donc de façon déclarative les conditions que chaque tuple x doit vérifier pour appartenir au résultat.

- x est une variable libre de $F(x)$.
- On introduit si besoin des variables liées par des quantificateurs $\exists$ ou $\forall$. Ces variables permettent par exemple de parcourir les relations, pour être comparées à x.

Exemples Calcul Relationnel

- Quel est le prenom et le nom de tous les étudiants
  - $\{x = (x_1, x_2) \mid \exists x' \in Etudiants((x_1, x_2) = x'[PRENOMETUD, NOMETUD])\}$
- Quel est le prenom et le nom des enseignants qui sont PR
  - $\{x = (x_1, x_2) \mid \exists x' \in Enseignants(x'[GRADE] =' PR' \wedge (x_1, x_2) = x'[PRENOMENS, NOMENS])\}$
- Quel est le nom des enseignants qui encadrent l'étudiant 53 ?
  - $\{x = (x_1) \mid \exists x' \in Encadre(x'[NUMETUD] = 107 \wedge \exists y' \in Enseignants(x'[NUMENS] = y'[NUMENS] \wedge y'[NOMENS] = x_1))\}$
- Quels est le num des étudiants qui n'ont pas d'encadrant.
  - $\{x = (x_1) \mid \exists x' \in Etudiants(x'[NUMETUD] = x_1 \wedge \forall y' \in Encadre(x'[NUMETUD] \neq y'[NUMETUD]))\}$
- Lister le nom de tous les étudiants et enseignants.
  - $\{x = (x_1) \mid \exists x' \in Etudiants(x'[NOMETUD] = x_1)\} \cup \{x = (x_1) \mid \exists x' \in Enseignants(x'[NOMENS] = x_1)\}$

Structured Query Langage - SQL

- Langage implémenté et universel d'interrogation d'une BD relationnel
- Traduit du calcul relationnel, donc déclaratif.
- Définit dans les années 80, dernière norme en 92
  - Beaucoup d'évolutions : UPSERT, Windows functions, Grouping Set, récursivité
- Se décompose en sous-ensembles :
  - DML : Manipulation (màj) et interrogation des données
  - DDL : Définition des données (au niveau du schéma)
  - DCL : Control des droits des utilisateurs
  - TCL : Control des transactions

Langage très puissant souvent sous-exploité

- Apprenez en pratiquant sur l'excellent site `https://www.pgexercises.com/`
- Si pouvez le faire en SQL, alors ne le faites pas autrement.

Exemples d'interrogations

- Quel est le prenom et le nom de tous les étudiants
  - SELECT PRENOMETUD, NOMETUD FROM Etudiants
- Quel est le prenom et le nom des enseignants qui sont PR
  - SELECT PRENOMENS, NOMENS FROM Enseignants WHERE GRADE='PR'
- Quel est le nom des enseignants qui encadrent l'étudiant 53 ?
  - SELECT NOMENS FROM Enseignants JOIN Encadre ON Enseignants.NUMENS=Encadre.NUMENS WHERE Encadre.NUMETUD=107
- Quels est le num des étudiants qui n'ont pas d'encadrant.
  - SELECT NUMETUD FROM Etudiants EXCEPT SELECT NUMETUD FROM Encadre
- Lister le nom de tous les étudiants et enseignants.
  - SELECT NUMETUD AS 'NOM' FROM Etudiants UNION SELECT NUMENS AS 'NOM' FROM Enseignants

Plan du chapitre

Définition des transactions

Transaction = ensemble de mises à jour

- Perçue comme atomique par l'utilisateur
- Il n'y a que deux états de fin possible
  - annulation : aucune trace, rien n'a été fait.
  - validation : modifications présentes et pérennes même en cas de panne
- modifications invisibles PENDANT la transaction.
  - Les requêtes concurrentes ne voient que l'état AVANT
- La base est cohérente AVANT et APRES la transaction
  - Contraintes pas forcément vérifiées PENDANT la transaction.

On résume par l'acronyme ACID : Atomicité, Cohérence, Isolation, Durabilité

Plan du cours

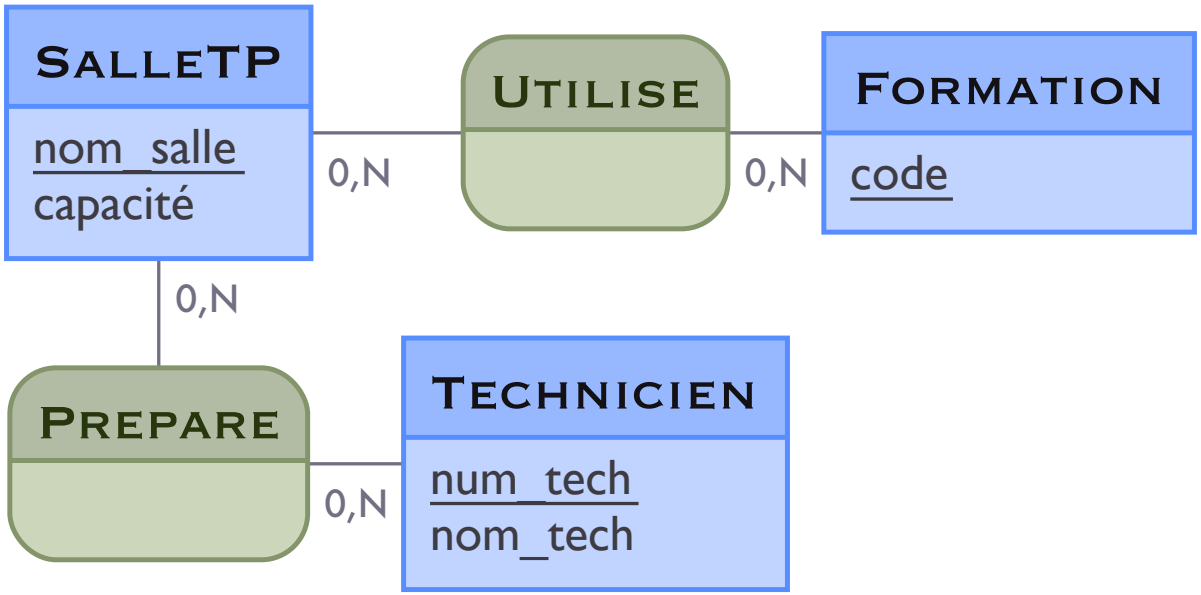
Plan du chapitre

Intuition

- Concevoir une BD relationnelle, c'est décider des attributs, des relations, des clés et clés étrangères qui traduisent le cahier des charges.
- le processus (EA => traduction relationnelle) est souvent mal maîtrisé, et ne fait pas tout
- De plus, il est important de savoir reconnaître des problèmes
  - De très nombreuses BD existantes sont mal conçues !
  - Comment le détecter ?
  - Comment le qualifier ?

Exemple de cahier des charges

Soit les données modélisées de la façon suivantes.



Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

- Un tableau n'est pas toujours une relation !

Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

- Un tableau n'est pas toujours une relation !
- On a deux fois la même ligne : hors relationnel

Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

- Un tableau n'est pas toujours une relation !
- On a deux fois la même ligne : hors relationnel
- des informations de types différents dans une même colonne : hors relationnel

Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

- Un tableau n'est pas toujours une relation !
- On a deux fois la même ligne : hors relationnel
- des informations de types différents dans une même colonne : hors relationnel
- Des cases avec des listes : hors relationnel

Les "fausses" BD relationnelles : exemple

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1,2	Smith, James
	TP3	17 (un ordinateur de fonctionne pas)	2	James
	TP2	18	3 (depuis le 1/1/2023)	Scott
	TP2	18	1,2	Smith, James

TABLE – Exemple de BD non relationnelle

- Un tableau n'est pas toujours une relation !
- On a deux fois la même ligne : hors relationnel
- des informations de types différents dans une même colonne : hors relationnel
- Des cases avec des listes : hors relationnel
- => Non utilisable, non maintenable.

Les BD relationnelles mal conçues

En respectant cette fois le modèle relationnel, on propose la relation suivante.

salles	nom_salle	capacite	num_tech	nom_tech
	TP2	18	1	Smith
	TP2	18	2	James
	TP3	17	2	James
	TP4	20	NULL	NULL

TABLE – Exemple de relation mal conçue

Il y a de la redondance !

- On répète plusieurs fois la capacité d'une même salle, le nom d'un technicien
  - Mises à jour risquées, et coûteuses
- Quel est le sens de la valeur NULL ? Si on supprime la dernière ligne, on perd la salle ?

Plan du chapitre

Comment détecter la redondance ?

La redondance est une conséquence du cahier des charges

- Un même technicien a toujours le même nom. . .
- . . . et on identifie un technicien avec son numéro.
- Donc pour un numéro donné, on ne peut avoir qu'un seul nom.
- On modélise cette info par une dépendance fonctionnelle (DF)
  - On note num_tech $\rightarrow$ nom_tech (on lit la flèche "détermine")
  - En exprimant cette DF, on "formalise" des redondances éventuelles
  - Quelles autres DF a-t-on dans cet exemple ?

Définition des dépendances fonctionnelles

Syntaxe

- Expression $X \rightarrow Y$
- Avec X et Y des ensembles d'attributs

Sémantique

- On dit que r satisfait $X \rightarrow Y$
- $\forall t_1, t_2 \in r, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$
- Remarque : Si $Y \subseteq X$, la DF est dite triviale, car toujours satisfaite

DF valides, modèle et conséquence sémantique

Soit Σ et Σ' des ensembles de DF exprimées sur un schéma R.

- $r \models X \rightarrow Y$ signifie r satisfait, où "est un modèle de" $X \rightarrow Y$
- $r \models \Sigma$ signifie $\forall f \in \Sigma, r \models f$
- $\Sigma \models \Sigma'$ signifie $r \models \Sigma \implies r \models \Sigma'$
  - On dit que Σ' est une conséquence sémantique de Σ.

On considère dans la suite des couples (R, Σ), où R est un schéma de relation et Σ un ensemble de DF valides sur R[3]

Une relation r définie sur (R, Σ) sera telle que son schéma est R et que $r \models \Sigma$.

3. Traduisent des contraintes du cahier des charges.

Lien avec la contrainte de clé

La contrainte de clé s'eprime par les DF

- X est une clé dans (R, Σ) ssi $X \rightarrow R$ peut être déduite [a] dans Σ
- Un doublon sur X impliquerait un doublon de tuple => impossible
- Donc les trois affirmations sont équivalentes :
  - X est une clé dans (R, Σ)
  - $X \rightarrow R$ est valide à partir de Σ
  - Les doublons sur les valeurs de X sont interdits dans toutes les relations définies sur (R, Σ)

a. cette notion sera explicitée plus loin

Plan du chapitre

Des clés, que des clés

- Toute $DF\ X \rightarrow Y$ non triviale crée de la redondance si elle est valide
- Car en cas de répétition de X, il faudra répéter Y
- Sauf si X est clé : la répétition est alors impossible

Que faire alors si $X \rightarrow Y$ est valide dans un cahier des charges ?

- Bien sûr on n'abandonne pas cette contrainte pour autant !
- On va chercher à "séparer" les attributs du schéma en plusieurs schémas, de façon à ce que X soit une clé partout où X et Y sont réunis
- Cela s'appelle normaliser une relation par décomposition du schéma. Les informations d'origine seront retrouvées grâce à des jointures.

Les niveaux de normalisation

Soit un schéma (R, Σ). On définit les "niveaux de qualité" suivants :

- Forme Normale de Boyce-Codd (FNBC)
  - Pour toute DF valide $X \rightarrow Y$ non triviale dans Σ, X est une clé (pas forcément minimale)
    –> Plus aucune redondance générée par les DFs, on dit que toutes les DF sont des conséquences des clés.
- Troisième Forme Normale (3FN)
  - Accepte certaines DF $X \rightarrow A$ non triviales même si X non clé
  - MAIS uniquement si A fait partie d'une clé minimale de (R, Σ)
- Deuxième Forme Normale : historique, pas d'intérêt en pratique
- Première Forme Normale : les domaines des attributs sont atomiques.

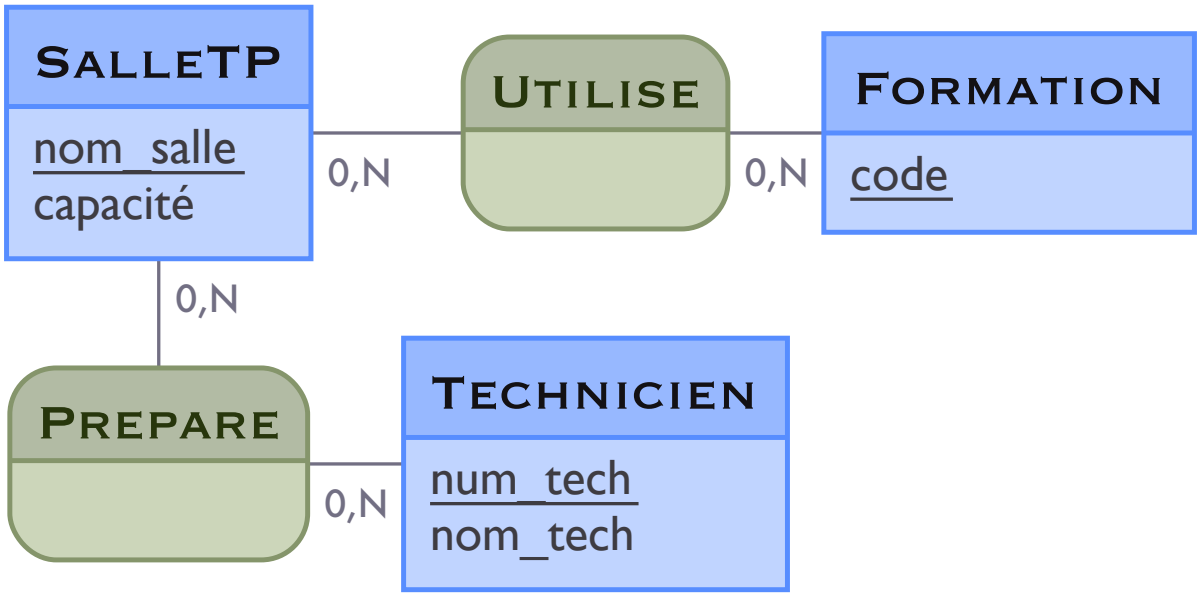
les niveaux de normalisation (suite)

FNBC $\subseteq$ 3FN $\subseteq$ 1FN

- Toute relation qui n'est pas en FNBC est dite "dénormalisée"
  - Elle doit être corrigée si on compte faire des mises à jour !
  - Des algorithmes de correction seront vus au chapitre suivant
- Mais les DF ne capturent pas toutes les redondances...
  - Les Dépendances de Jointures

Plan du chapitre

Rappel du schéma



Dépendance de jointure : intuition (1)

Toujours à partir du même schéma E/A, on se sert de la relation suivante pour traduire à la fois l'association "Utilise" et l'association "Prépare".

salles	num_salle (S)	code (C)	num_tech (T)
	TP1	L3IF	SMITH
	TP1	M2TIW	JAMES
	TP2	L3IF	SCOTT
	TP2	L3IF	JAMES

Cette relation est bien en FNBC (aucune DF non triviale valide)

- Pourtant les problèmes ne manquent pas !
- Tuples 1 et 2 : on pourrait intervertir "SMITH" et "JAMES" -> choix arbitraire ?
- Résultats de requêtes différents selon le choix...

Dépendance de jointure : intuition (2)

salles	num_salle (S)	code (C)	num_tech (T)
	TP1	L3IF	SMITH
	TP1	M2TIW	JAMES
	TP2	L3IF	SCOTT
	TP2	L3IF	JAMES

D'où vient le problème ?

- Le cahier des charges ne définit pas de relation directe entre les salles et les techniciens.
- "Si une salle s est en lien avec une formation c et un technicien t, alors le tuple $< s, c, t >$ est pertinent et devrait exister.
- On note : $\bowtie [SC, ST]$

Dépendance de jointure : intuition (3)

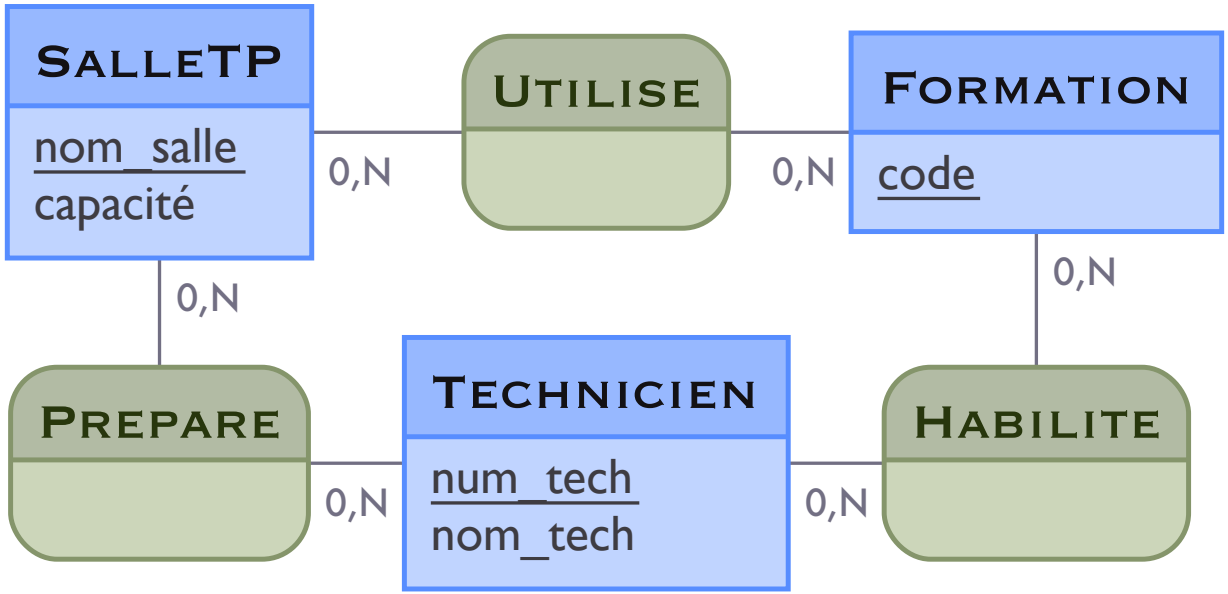
salles	SALLE (S)	FORMATION (F)	TECHNICIEN (T)
	TP1	L3IF	SMITH
	TP1	M2TIW	JAMES
	TP1	M2TIW	SMITH
	TP1	L3IF	JAMES
	TP2	L3IF	SCOTT
	TP2	L3IF	JAMES

Ici la relation respecte bien cette contrainte, tous les tuples pertinents sont bien présents.

Mais du coup, cette contrainte entraîne une forte redondance, elle caractérise une mauvaise conception.

Autre exemple (1)

Modifions un peu le cahier des charges : les techniciens sont maintenant habilités (ou pas) à travailler avec certaines formations.



Autre exemple

Reprenons la relation d'origine.

salles	num_salle (S)	code (C)	num_tech (T)
	TP1	L3IF	SMITH
	TP1	M2TIW	JAMES
	TP2	L3IF	SCOTT
	TP2	L3IF	JAMES

Le cahier des charges ne définit pas de relation ternaire entre S, C et T. Il nous faut donc une contrainte du type : "Si une salle s est utilisée par une formation c, que cette salle est préparée par le technicien t, et que le technicien est habilité pour la formation c, alors le tuple $< s, f, t >$ est pertinent et doit exister.
On exprime cette contrainte par $\bowtie \{SC, ST, CT\}$. Est-elle respectée ici ?

Autre exemple (2)

salles	num_salle (S)	code (C)	num_tech (T)
	TP1	L3IF	SMITH
	TP1	M2TIW	JAMES
	TP1	L3IF	JAMES
	TP1	L3IF	SCOTT
	TP2	L3IF	JAMES

Il ne manque qu'un tuple ici ; notons que "SMITH" n'est pas habilité pour "M2TIW" donc on ne doit pas ajouter le tuple $< TP1, M2TIW, SMITH >$.

Dépendances de jointure : définition

Syntaxe

- Expression $\bowtie \{X_1, ...X_n\}$
- $X_1 \cup ... \cup X_n = R$
- chaque X_i est appelé une composante de jointure.

Satisfaction par une relation r

- $r \models\!\!\bowtie \{X_1, ...X_n\} \iff r = \pi_{X_1}(r) \bowtie ... \bowtie \pi_{X_n}(r)$
- Ce qui peut s'écrire aussi : $\forall t_1, ..., t_n \in r, \forall t$:
  $t[R_1] = t_1[R_1] \wedge \ldots t[R_n] = t_n[R_n] \implies t \in r$

De façon informelle : Si un tuple se trouve dans la jointure sur les composantes de jointure, il DOIT exister dans la relation.

Dépendances de jointure : définition

Remarques

- Si l'une des composantes de jointure est R, la DJ est triviale
- $r \models X \rightarrow Y \Rightarrow r \models \bowtie \{XY, X(R-Y)\}$

Niveaux de normalisation supplémentaires

Ici encore, on souhaite que les contraintes soient garanties par les clés, rien que les clés. Soit Σ un ensemble de DF et de DJ.

- un schéma (R, Σ) est en 4ème Forme Normale s'il est en FNBC et si pour toute DJ non triviale de la forme $\bowtie [XY, XZ]$, X est clé.
- un schéma (R, Σ) est en 5ème Forme Normale s'il est en FNBC et pour toute DJ non triviale $\bowtie [X_1, \ldots, X_n]$, celle-ci est "la conséquence des clés".

Que signifie qu'une DJ est une conséquence des clés ?

Soit (R, Σ) avec Σ un ensemble de DF et de DJ. Soit une DJ $j = \bowtie \{X_1, \ldots, X_n\}$ une des dépendances de Σ.
On dit que j est "une conséquence des clés", ou "est à base de clés" si, pour toute relation r :
"r respecte les clés issues de Σ" $\Longrightarrow$ $r \models j$
Pour le tester, il faut appliquer le petit algorithme suivant :

- Si $X_i \cap X_j$ est une clé de (R, Σ), on les remplace par $X_i \cup X_j$
- On répète cette opération tant que cela est possible.
- Si on finit par obtenir R comme composante de jointure : j est à base de clés
- Sinon, j n'est pas à base de clés, donc (R, Σ) n'est pas normalisé.

Implication entre formes normales

Chaque forme normale est un "raffinement" de la précédente. Ainsi, si (R, Σ) est un schéma de relation muni d'un ensemble de DF et DJ, on a :

$$\text{R en 5FN} \Rightarrow \text{R en 4FN} \Rightarrow \text{R en FNBC} \Rightarrow \text{R en 3FN} \Rightarrow \text{R en 1FN}$$

Intuition : supposons que chaque DJ valide est une conséquence des clés (5FN respectée). Soit $X \rightarrow Y$ une DF valide quelconque, alors la DJ $\bowtie \{XY, XZ\}$ (avec $Z = R - X - Y$) est valide ; donc X est clé. Donc la FNBC est respectée.

Que faire si un schéma redondant ?

- Comprendre la situation sur un diagramme E/A
- Traduire ce diagramme en relationnel
- Le résultat est toujours en 5FN
- MAIS
  - L'ajout des contraintes peut créer des problèmes
    - Contraintes X, T, XT, I
    - Contraintes du cahier des charges non modélisées en E/A
- Si les problèmes persistent, l'analyse doit être poussée...
- CF suite du cours.

Plan du chapitre

Exercices (1)

Dites si la relation suivante respecte les contraintes

- $AB \rightarrow C$; $AE \rightarrow BC$; $DE \rightarrow R$
- $\bowtie \{AD; ABCE\}$

R	A	B	C	D	E
	1	2	3	4	5
	1	2	3	4	6
	1	2	3	7	5
	1	2	3	7	6

TABLE – Exemple de relation

Exercices (2)

- Soit le schéma conceptuel donné dans la diapositive 55. Exprimez des contraintes de dépendances fonctionnelles (DF) non triviales qui sont valides dans ce cahier des charges.
- En justifiant par ces DF, dites si les relations suivantes sont en Forme Normale de Boyce-Codd (FNBC). Si ce n'est pas le cas, trouvez un exemple de relation qui fait apparaître une redondance liée à une DF valide non triviale.

$$Commandes(date\_commande, num\_client, ville\_client)$$

$$Commandes(date\_commande, num\_client, adresse\_commande, code\_produit)$$

$$Produits(code\_produit, designation\_produit, code\_categorie)$$

$$Commandes(date\_commande, num\_client, code\_produit, code\_categorie)$$

- Pour la dernière relation exhibez une redondance qui n'est pas liée à une DF.

Exercices (3)

- Pour les schémas relationnels suivants, dites si les relations sont en 5FN. Si ce n'est pas le cas, essayez de dire si elles sont en 3FN.

$$R(ABCDEF)$$

$$(R = ABCDEF, \{AB \rightarrow C; C \rightarrow DEF; \bowtie \{ADF; ABCDE\}\})$$

$$(R = ABCDEF, \{AB \rightarrow C; C \rightarrow ADEF\})$$

$$(R = ABCDEF, \{AB \rightarrow C; C \rightarrow ABDEF; \bowtie \{ADF; ABCDE\}\})$$

Plan du cours

Plan du chapitre

Les dépendances cachées...

La conception est guidée par les contraintes

- Pour éviter la redondances, la cohérence entre les relations
- Les contraintes (DF, DJ, ...) sont déduites du cahier des charges, *mais* :
  - Elles peuvent être très nombreuses, plusieurs centaines
  - Elles peuvent avoir des interactions
- On doit se doter de quelques formalismes et algorithmes

Les dépendances cachées... (suite)

Trois types de contraintes sont utilisées

- A l'intérieur d'un schéma de relation :
  - Les Dépendances fonctionnelles (DF)
  - Les dépendances de Jointure (DJ)
- Entre plusieurs schémas de relation :
- Les Dépendances d'Inclusions (DI)

Plan du chapitre

Les Dépendances d'Inclusion (DI)

Syntaxe

- $R_1[X] \subseteq R_2[Y]$ avec R_1 et R_2 des schémas de relations
- X et Y sont des séquences d'attributs de même taille.
  - Donc l'ordre des attributs est important !

Sémantique

- Soit deux relations r_1 et r_2 définies sur R_1 et R_2
- $\{r_1; r_2\} \models R_1[X] \subseteq R_2[Y] \Longleftrightarrow \forall t_1 \in r_1 \exists t_2 \in r_2 | t_1[X] = t_2[Y]$
- De façon équivalente :
  $\{r_1; r_2\} \models R_1[X] \subseteq R_2[Y] \Longleftrightarrow \Pi_X(r_1) \subseteq \Pi_Y(r_2)$
  –> Chaque valeur que prend la séquence d'attributs X dans r_1 doit être une valeur prise par la séquence d'attributs Y dans r_2

Les Dépendances d'Inclusion (DI) (suite)

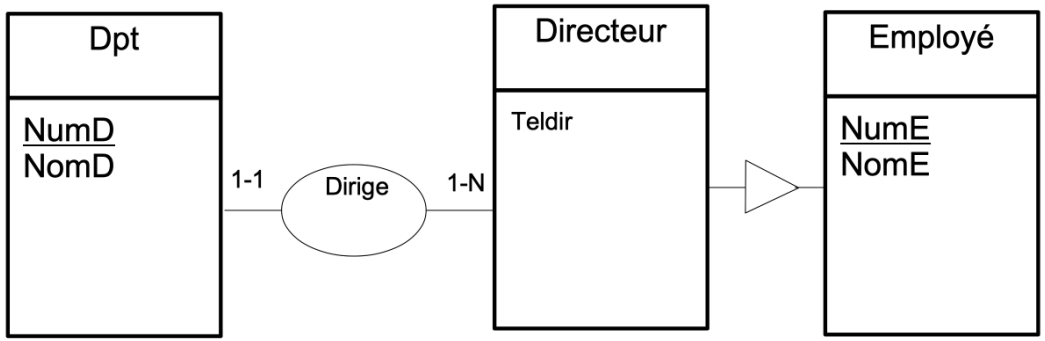
Cas particulier : les références

- La partie droite de la DI est parfois une clé de S
- Ainsi la DI permet de *faire référence* à un tuple unique de s
- La DI est alors appelée *contrainte d'intégrité référentielle*
- Dans ce cas, la partie gauche est une clé étrangère de R

Les dépendances triviales

- Une DI est triviale si elle est de la forme $R[X] \subseteq R[X]$
- Forcément toujours satisfaite

Les Dépendances d'Inclusion (DI) (exemple)



Dpt	NumD	NomD	NumE_D
	D1	Info	23
	D2	Maths	2
	D3	Chimie	2

Dir	NumE_D	TelD
	2	06..
	23	07..

Emp	NumE	NomE
	2	SMITH
	5	JAMES
	23	SCOTT

Les DI traduisent les liens entre entités

- $Dpt[NumE\_D] \subseteq Dir[NumE\_D]$ (Intégrité Référencielle)
- $Dir[NumE\_D] \subseteq Emp[NumE]$ (intégrité Référentielle)
- $Dir[NumE\_D] \subseteq Dpt[NumE\_D]$ (DI non référentielle, traduit la participation obligatoire)

Plan du chapitre

L'inférence de dépendances

Des dépendances "cachées"

- Lorsque des dépendances sont valides, elles peuvent engendrer d'autres dépendances forcément valides
- Par exemple, par transitivité dans les DF.
- Cela a de l'influence sur la conception (repérer la redondance, trouver les clés minimales...) !
- Comment être certain qu'on ne rate rien, surtout dans des relations de grande taille avec beaucoup de dépendances ?

L'inférence de dépendances (suite)

Définition formelle du problème d'inférence

- Soit D un schéma de BD muni d'un ensemble de dépendances Σ
  - Cela veut dire que $\forall d$ sur D, on a $d \models \Sigma$
- Soit σ une dépendance quelconque exprimée sur D
- Problème de l'inférence : Soit d sur D, $d \models \Sigma \Rightarrow d \models \sigma$?
- Si la réponse est oui, cela s'exprime en écrivant : $\Sigma \models \sigma$

C'est donc un problème de *décision* : réponse oui ou non

- Sa complexité va dépendre des types de dépendances.

On s'intéresse aussi au problème *d'énumération* associé

- Etant donné Σ, quelles sont *toutes* les dépendances σ telles que $\Sigma \models \sigma$?

Comment démontrer l'inférence ?

Pour démontrer que $\Sigma \models \sigma$

- Preuve sémantique
  - On suppose une BD $d \models \Sigma$, on démontre $d \models \sigma$
  - En utilisant les définitions de satisfaction.
  - "A la main", complexe à automatiser et à vérifier...
- Déduction syntaxique
  - A l'aide d'un système syntaxique de génération de preuves
  - Permet la **vérification** de la preuve
  - Pour les DF : engendre un algorithme simple
- L'algorithme de la poursuite
  - "Automatisation" des preuves sémantiques
  - Sorte de "démonstration par l'exemple", sans perte de généralité.

Les règles de déduction syntaxique

Règles de "génération" des contraintes

- Expressions syntaxiques
- De la forme : Si Σ Alors σ
- Permettent de faire des preuves logiques, de la dérivation
- On note $\Sigma \vdash \sigma$ si on peut dériver syntaxiquement σ à partir de Σ
- Exemple : transitivité dans les DF $\{X \rightarrow Y; Y \rightarrow Z\} \vdash X \rightarrow Z$

Un système d'inférence est un ensemble de règles d'inférence.

Les règles de déduction syntaxique (suite)

Règles correcte

- Une règle est dite *correcte*, si elle est "cohérente" avec la conséquence sémantique :
- $\Sigma \vdash \sigma \Rightarrow \Sigma \models \sigma$
- Exemple :
  - La transitivé dans les DF est correcte
  - La règle : $\{X \rightarrow Y\} \vdash X \rightarrow Z$ est bien une règle d'inférence, mais n'est pas correcte car on pourra trouver de nombreux contre-exemples !

Ensemble complet de règles

- un système d'inférence est dit *complet* s'il permet de dériver TOUTES les dépendances qu'il est possible de dériver.
- Dans ce cas : $\Sigma \vdash \sigma \Leftarrow \Sigma \models \sigma$

Les règles de déduction syntaxique (suite et fin)

Système correct ET complet

- Un système d'inférences est *correct et complet*, si toutes ses règles sont correctes, et qu'il est complet.
- Il permet de capturer tous les cas, l'ensemble de la sémantique potentiellement "cachée"
- Malheureusement, il n'en existe pas pour tous les types de dépendances. . .
- Et l'existence d'une système juste et complet ne veut pas dire que le problème de l'inférence est résolu :
  - Peut-on tout faire avec des preuves finies ? de taille polynomiale ?

Système d'inférence pour les DF

Les règles suivantes, qui se restreignent aux DF, sont toutes correctes.

$$\frac{Y \subseteq X}{X \to Y}$$ (réflexivité)

$$\frac{X \to Y \qquad X \to Z}{X \to YZ}$$ Union

$$\frac{X \to Y}{WX \to WY}$$ (augmentation)

$$\frac{X \to YZ}{X \to Y}$$ décomposition

$$\frac{X \to Y \qquad Y \to Z}{X \to Z}$$ (transitivité)

$$\frac{X \to Y \qquad WY \to Z}{WX \to Z}$$ (pseudo-transitivité)

Plusieurs systèmes d'inférence justes et complets, par exemple :

- {Réflexivité, (Augmentation ou Union), Transitivité}
  - C'est le système d'Armstrong
- {Réflexivité, Pseudo-transitivité}

Système d'inférence pour les DF (suite)

Trouver des preuves est difficile !

- En présence de beaucoup de DF, cela peut être très complexe
- On peut *ne pas arriver* à prouver une inférence...
- ... mais comment être sûr que ce n'est *pas possible* ?
- Il est fastidieux d'automatiser la construction de preuves par un algorithme.

Même si ces règles sont précieuses pour comprendre les mécanismes d'inférence, elles restent très peu utilisables en pratique, par exemple pour trouver les clés minimales, ou pour être certain de ne pas "rater" de DF.

Un opérateur de fermeture à la rescousse

Opérateur de fermeture dans les DF

- Soit R un shéma et Σ un ensemble de DF.
- On note $X^+ = \{A \in R | \Sigma \models X \rightarrow A\}$
- Intuition : X^+ est l'ensemble de tous les attributs qui sont déterminés par X par les DF de Σ.

Un algorithme simple pour le calcul de X^+

1. Initisalisation : $X^+ = X$ (en effet, la df $X \rightarrow X$ est triviale)
2. Pour chaque DF de Σ, si sa partie gauche est contenue dans X^+, alors on rajoute sa partie droite dans X^+.
3. Si on a parcouru toutes les DF sans faire aucun changement, alors c'est fini.
4. Sinon, on recommence à l'étape 2.

Exemple de calcul de fermeture

Soit $R = ABCDE$ et $\Sigma = \{AC \rightarrow D; E \rightarrow C; A \rightarrow E\}$

On veut calculer AB^+

- Initialisation : $AB^+ = AB$
- On parcours les DF, on voit que la partie gauche de $A \rightarrow E$ est dans AB. Donc on peut rajouter E pour obtenir $AB^+ = ABE$
- On voit maintenant qu'on peut 'déclencher' la DF $E \rightarrow C$, donc on rajoute C pour obtenir $AB+ = ABCE$
- Ce qui permet d'utiliser $AC \rightarrow D$ pour obtenir $AB^+ = ABCDE$

On peut vérifier en exercice que $A^+ = ACDE$, $E^+ = CE$, $AE^+ = ACDE$.

Remarques complémentaires sur la fermeture

Quand utiliser cet algorithme ?

1. Pour savoir si X est une clé, on regarde si $X^+ = R$
2. Pour savoir si $\Sigma \models X \rightarrow Y$, on regarde si $Y \subseteq X^+$
3. ... on verra d'autres utilisations.

Propriétés

1. L'algorithme a une complexité linéaire. Le problème d'inférence dans les DF est donc facile !
2. La fonction $.^+$ a des propriétés intéressantes :
   - $X \subseteq X^+$ (extensive)
   - Si $X \subseteq Y$ Alors $X^+ \subseteq Y^+$ (croissante)
   - $(X^+)^+ = X^+$ (idempotente)

Système d'inférence des dépendances de jointure de taille 2

Rappels du chapitre précédent

- Soit R un schéma de relation
- Une DJ de taille 2 sur R est une expression de la forme $\bowtie \{XY, XZ\}$ avec $Z = R - X - Y$
- Autrement dit, l'union des deux composantes de jointure est toujours R, et on nomme X leur intersection.
  - indiquant que les tuples de la relation sont des conséquences des paires XY et des paires XZ.
- $r \models \bowtie [\{XY, XZ\}$ ssi $r = \Pi_{XY}(r) \bowtie \Pi_{XZ}(r)$
- Dit autrement : si deux tuples de r sont égaux sur X, alors si on échange leurs valeurs sur Y et Z, on retrouve des tuples de r.

Inférence des DJ de taille 2 (suite)

Exemple

- Un enseignant (E) a plusieurs bureaux (B), et donne plusieurs cours (C). Les cours qu'il donne sont indépendants de la localisation de ses bureaux.
- $R(E, B, C)$. On voit ici que les couples (E, B) sont décrits par le cahier des charges, ainsi que les couples (E, C) ; en revanche les couples (B, C) ne sont qu'une conséquence des deux autres.
- Ceci est capturé par la DJ $\bowtie \{EB, EC\}$
- Ainsi, si e_1 donne un cours c_1, et que e_1 a le bureau b_1, alors le triplet (e_1, b_1, c_1) doit exister dans la relation.

Règles d'inférence des DJ de taille 2

Le système suivant est correct et complet pour les DJ de taille 2 définies sur un schéma de relation R. La notation "-" doit être comprise comme étant l'ensemble des attributs qui ne figurent pas encore dans la DJ.

- Réflexivité

$$\frac{}{\bowtie \{X, R\}}$$

- Permutation

$$\frac{\bowtie \{X_1, X_2\}}{\bowtie \{X_2, X_1\}}$$

- pseudo-transitivité

$$\frac{\bowtie \{XY, X-\} \qquad \bowtie \{YY', Y-\}}{\bowtie \{X(Y' \setminus Y), X-\}}$$

- Augmentation

$$\frac{\bowtie \{XY, X-\}}{\bowtie \{XYZ, XZ-\}}$$

Interactions entre DF et DJ

Les règles suivantes capturent les interactions entre les DF et les DJ de taille 2.

- Conversion

$$\frac{X \rightarrow Y}{\bowtie \{XY, X-\}}$$

- Interaction

$$\frac{\bowtie \{XY, X-\} \qquad XY \rightarrow Z}{X \rightarrow Z - Y}$$

- La règle de conversion rappelle simplement que si une DF est satisfaite, alors une DJ induite de taille 2 est satisfaite.

Systèmes d'Inférence des DI

Les règles d'inférence suivantes sont appelées système d'inférence de Casanova *et al.* pour les DI, dans lequel σ est une permutation d'un sous-ensemble de $\{1..n\}$:

- Réflexivité

$$\frac{}{R[X] \subseteq R[X]}$$

- Permutation & projection

$$\frac{R[A_1 ... A_n] \subseteq S[B_1 ... B_n]}{R[A_{\sigma(1)} ... A_{\sigma(k)}] \subseteq S[B_{\sigma(1)} ... B_{\sigma(k)}]}$$

- Transitivité

$$\frac{R[X] \subseteq S[Y] \qquad S[Y] \subseteq T[Z]}{R[X] \subseteq T[Z]}$$

Interactions entre les DF et les DI

Il existe des interactions entre les DF et DI illustrées par les règles suivantes.

- Pullback

$$\frac{R[XY] \subseteq S[TU] \qquad S : T \to U}{R : X \to Y} \; |X| = |T|$$

- Collection

$$\frac{R[XY] \subseteq S[TU] \qquad R[XZ] \subseteq S[TV] \qquad S : T \to U}{R[XYZ] \subseteq S[TUV]} \; |X| = |T|$$

Ce système n'est pas complet ; il n'existe pas de système complet *fini* pour capturer les interactions entre DF et DI.

A la poursuite des dépendances

La méthode de la poursuite est un algorithme basé sur la définition de satisfaction des contraintes. Elle permet de démontrer par l'exemple que $\Sigma \models \sigma$, ou démontrer que c'est faux en exhibant du même coup un contre-exemple.

Principe

1. Initialiser une relation en situation de contredire la dépendance "chassée"
2. On utilise des valeurs fixes arbitraires, et des variables
3. On utilise les dépendances en hypothèse pour fixer des valeurs aux variables

L'algorithme de la poursuite (the chase algorithm)

ETAPE 1 : préparation de la poursuite

- On part d'une relation vide r sur R.
  - Si $\sigma = X \rightarrow Y$ est une DF :
    - On insère un tuple avec la valeur arbitraire 0 dans tous les attributs
    - le deuxième tuple est égal à 0 sur X pour créer un doublon sur la partie gauche. Pour chacun des autres attributs A, on affecte une variable x_A
  - Si $\sigma = \bowtie \{X_1, X_2, ..., X_n\}$ est une DJ :
    - On insère un tuple t_i pour chaque composante de jointure X_i $t_i[A] = 0$ si $A \in X_i$, $t_i[A] = x_{iA}$ si $A \notin X_i$

Préparation de la poursuite... Exemples.

Préparation pour démontrer la DF $A \rightarrow D$

r	A	B	C	D	E
t_1	0	0	0	0	0
t_2	0	x_B	x_C	x_D	x_E

Préparation pour démontrer la DJ $\bowtie [ABC; ADE]$

r	A	B	C	D	E
t_1	0	0	0	x_{1D}	x_{1E}
t_2	0	x_{2B}	x_{2C}	0	0

A la poursuite des dépendances

ETAPE 2 : la poursuite

- On "force" dans r chaque dépendance σ' de Σ :
  - Si $\sigma' = X' \rightarrow Y'$ est une DF
    - si deux tuples sont égaux sur X', on "force" leur égalité sur Y'
  - Si $\sigma' = \bowtie [XY, X(R-Y)]$ est une DJ de taille 2
    - Lorsque deux tuples sont égaux sur X, on intervertit leurs valeurs sur Y et $R-Y$. Si on obtient des nouveaux tuples, on les rajoute.

La poursuite converge toujours en un nombre fini d'étapes, c'est à dire qu'on obtient une relation dans laquelle les contraintes de Σ sont toutes satisfaites.

La poursuite, exemples.

Supposons que $A \rightarrow C \in Sigma$.

r	A	B	C	D	E
t_1	0	0	0	0	0
t_2	0	x_B	x_C	x_D	x_E

La DF $A \rightarrow D$ nous impose d'affecter la valeur 0 à x_C pour être satisfaite :

r	A	B	C	D	E
t_1	0	0	0	0	0
t_2	0	x_B	0	x_D	x_E

La poursuite, exemples.

Supposons maintenant que $\bowtie [ADE, ABCD] \in \Sigma$

r	A	B	C	D	E
t_1	0	0	0	0	0
t_2	0	x_B	x_C	0	x_E

Comme les deux tuples sont égaux sur AD, la contrainte $\bowtie [ADE, ABCD]$ ne sera satisfaite que si on ajoute deux tuples qui intervertissent les valeurs sur E et BC. On obtient deux nouveaux tuples.

r	A	B	C	D	E
t_1	0	0	0	0	0
t_2	0	x_B	x_C	0	x_E
t_3	0	0	0	0	x_E
t_4	0	x_B	x_C	0	0

A la poursuite des dépendances

ETAPE 3 : la conclusion

- Si on cherchait à prouver $\Sigma \models X \rightarrow Y$
  - si les attributs qui composent Y ne prennent que des valeurs 0, alors ($\Sigma \models X \rightarrow Y$)
  - sinon, on dispose d'un contre-exemple ($r \models \Sigma \wedge r \not\models X \rightarrow Y$)
- Si on cherchait à prouver la DJ $\bowtie [X_1, ..., X_n]$
  - Si au moins une ligne est entièrement égale à 0, alors $\Sigma \models \bowtie [X_1, ..., X_n]$)
  - sinon, on dispose d'un contre-exemple.

Bilan sur l'inférence

Le problème de l'inférence est décidable dans tous les cas pour les classes de contraintes étudiées (DF, DJ, DI). Mais il sa complexité est variable selon les cas :

- Pour les DF
  - Plusieurs systèmes corrects et complets, inférence linéaire (fermeture ou poursuite)
- Pour les DJ de taille 2
  - Plusieurs systèmes corrects et complets, problème de l'inférence linéaire (poursuite)
  - Système complet pour les DF + DJ de taille 2. Inférence linéaire (poursuite)
- Pour les DJ de taille > 2
  - Pas de système d'inférence.
  - Inférence NP-Complet pour prouver une DJ >2 à partir de DF et DJ >2 (la poursuite termine mais peut avoir une taille exponentielle)

Bilan sur l'inférence (suite)

- Pour les DI [4]
  - Système d'inférence juste et complet, mais inférence pspace-complet dans le cas général.
  - pas de système complet pour DI + DF, inférence indécidable dans le cas général
- Pour les DI sans cycles [5]
  - Problème de l'inférence NP-complet
  - Problème de l'inférence exponentiel pour les DF et DI sans cycle

4. Notez qu'une variante de la poursuite, non vue ici, existe pour les DI
5. Un cycle est de la forme $R[X] \subseteq \ldots \subseteq R[Y]$

Plan du chapitre

Qu'est ce qu'une couverture ?

Equivalence entre les ensembles de dépendances

- On a vu qu'à partir de dépendances connues, on pouvait en dériver d'autres
- Il y a donc plusieurs façons de représenter la même sémantique
  - Donc traduire en contraintes le même cahier des charge

Définition

- Soient Σ_1 et Σ_2 deux ensembles de dépendances
- On dit Σ_1 est une couverture de Σ_2 si
  - $\Sigma_1 \models \Sigma_2$ ET $\Sigma_2 \models \Sigma_1$
- Donc la relation de couverture est une relation d'équivalence

Couvertures pour les DF

On définit des couvertures avec des propriétés diverses

Soit Σ un ensemble de dépendances et Σ' une couverture de Σ.

- Σ' est non redondante si :
  - $\forall \Sigma'' \mid \Sigma'' \models \Sigma, \Sigma'' \not\subset \Sigma'$
  - Donc on ne peut retirer aucune dépendance à Σ' sans perte de sémantique
- Σ' est minimale si :
  - $\forall \Sigma'' \models \Sigma, |\Sigma''| \geq |\Sigma'|$
  - Donc Σ' utilise un nombre minimal de dépendances pour exprimer la sémantique
- Σ' est optimale si :
  - Σ' utilise un nombre optimal de "signes" pour exprimer la sémantique

Si Σ' est optimale, alors elle est minimale. Si Σ est minimale, alors elle est non redondante.

Algorithme : couverture minimale pour les DF

Data: Σ un ensemble de DF
Result: Σ' une couverture minimale de Σ
$\Sigma' := \emptyset$
for $X \rightarrow Y \in \Sigma$ **do**
  $\Sigma' := \Sigma' \cup \{X \rightarrow X^+\}$
end
for $X \rightarrow X^+ \in \Sigma'$ **do**
  if $\Sigma' - \{X \rightarrow X^+\} \vdash X \rightarrow X^+$ **then**
    $\Sigma' := \Sigma' - \{X \rightarrow X^+\}$
  end
end
return Σ'

- Cet algorithme est polynomial dans le nombre de DF et d'attributs dans Σ.
- La couverture minimale calculée par l'algorithme n'est pas unique : d'autres couvertures peuvent avoir le même nombre de DF, mais être différentes.
- Parmi celles-ci, certaines sont optimales ; malheureusement, leur calcul est un problème difficile dans le cas général (NP-Complet).
- On peut néanmoins réduire une couverture minimale en parcourant un à un chaque attribut et en le supprimant si cette suppression ne modifie pas la sémantique.

Réduction du nombre d'attributs pour un ensemble de DF

Data: Σ un ensemble de DF sur R.

$Min := \Sigma$

/* Réduction des parties gauches */

for $X \rightarrow Y \in Min$ **do**

    $W := X$

    for $A \in X$ **do**

        if $Min \models (W - A) \rightarrow X$ **then** $W := W - \{A\}$;

    end

    $Min := (Min - \{X \rightarrow Y\}) \cup \{W \rightarrow Y\}$

end

/* Réduction des parties droites */

for $X \rightarrow Y \in Min$ **do**

    $W := Y$

    for $A \in Y$ **do**

        $Min' := (Min - \{X \rightarrow Y\}) \cup \{X \rightarrow (W - A)\}$

        if $Min' \models X \rightarrow Y$ **then** $W := W - \{A\}$;

    end

    $Min := (Min - \{X \rightarrow Y\}) \cup \{X \rightarrow W\}$

end

Exemple de réduction

Exemple de réduction

Soit l'ensemble de DFs Σ :

$$\Sigma = AB \rightarrow ABCDF;\ B \rightarrow BCD;\ DE \rightarrow F;\ E \rightarrow D$$

Couverture minimale (à vérifier) Σ' :
$\{AB \rightarrow ABCDF;\ B \rightarrow BCD;\ E \rightarrow DEF\}$

- Réduction des parties gauches :

$$\{Min = AB \rightarrow ABCDF;\ B \rightarrow BCD;\ E \rightarrow DEF\}$$

- Réduction des parties droites :

$$Min = AB \rightarrow F;\ B \rightarrow CD;\ E \rightarrow DF$$

Plan du chapitre

BILAN : Ce qu'il faut savoir faire

- Comprendre les règles d'inférence pour les DF, DJ de taille 2, DI.
  - Savoir les appliquer
- Vérifier si X est une clé dans sa relation par le calcul de sa fermeture.
  - X clé de R ssi $X^+ = R$
- Résoudre l'inférence dans les DF par le calcul de la fermeture
  - Si Σ n'est composé que de DF, $\Sigma \models X \rightarrow Y$ ssi $Y \subseteq X^+$
- Résoudre une inférence de DJ ou DF par l'algorithme de la poursuite.
- Calculer une couverture minimale pour les DF, réduire les parties gauches et droites.

Exercices (1)

Soit $R = ABCDEF$ muni de l'ensemble Σ de DF :
$AB \rightarrow C$; $ACD \rightarrow B$;$C \rightarrow ABD$; $D \rightarrow E$; $CE \rightarrow A$; $BC \rightarrow D$;
$ABE \rightarrow C$

- Calculer les fermetures suivantes : D^+, AE^+, BD^+, AC^+
- Calculer toutes les clés de (R, Σ)

Exercices (2)

- Montrer que la pseudo-transitivité et la réflexivité forment ensemble un système d'inférence correct et complet pour les DF.
- Soient les deux ensembles de DF : $\Sigma = \{AB \rightarrow C; \bowtie [AB, AC]\}$ et $\Sigma' = \{A \rightarrow C\}$. Montrer que ces deux ensembles sont équivalents.

Exercices (3)

Les implications suivantes sont des illustrations de diverses règles d'inférence. Identifiez-les et démontrez-les par la technique de la poursuite (on suppose un schéma $R = ABCDEF$) :

- $\{AB \rightarrow C; CE \rightarrow D\} \models ABE \rightarrow D$
- $AB \rightarrow D \models\!\!\bowtie \{ABD; ABCEF\}$
- $\bowtie \{ABD; ABCEF\} \models\!\!\bowtie \{ABCEF, ABD\}$
- $\{\bowtie \{AE; ABCDF\}; \bowtie \{CDE; ABEF\}\} \models\!\!\bowtie \{ACD, ABEF\}$
- $\{\bowtie \{AE; ABCDF\}; AE \rightarrow D\} \models A \rightarrow D$

Plan du cours

Plan du chapitre

Une compétences avancées

Pourquoi savoir normaliser ?

- Le Modèle E/A est une aide
  - Il est parfois complexe de traduire le cahier des charges
  - Des contraintes peuvent ne pas être représentées
  - Parfois, il est précipité ou carrément absent.
- Il faut savoir qualifier le résultat : les formes normales.
- Et résoudre d'éventuels problèmes : la normalisation

Allers/retours entre équipe de conception et équipe Base de Données.
Ou bien c'est la même personne !

Plan du chapitre

Diviser pour régner

On décompose les schémas de relation en plusieurs schémas.

Pourquoi ?

- une relation dénormalisée ==> plusieurs relations en 5FN
- La redondance est évacuée, et avec elle les anomalies de MAJ
- La BD est plus compréhensible, plus facile à manipuler
- On se limite au maximum à des clés et clés étrangères
- On réduit la quantité de valeurs NULL

Comment ?

- En s'appuyant sur les DF et les DJ, donc les sources de redondance

Une décomposition guidée par les Dépendances de Jointure

RAPPEL : la satisfaction d'une DF implique celle d'une DJ particulière.

- $r \models X \rightarrow Y \Longrightarrow r \models\!\!\bowtie \{XY, X(R - X - Y)\}$

RAPPEL : La satisfaction d'une DJ indique une décomposition possible

- $r \models\!\!\bowtie [R_1, R_2] \Longleftrightarrow r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$

Une DJ satisfaite est une condition nécéssaire et suffisante pour une décomposition sans perte de données.

Une décomposition guidée par les Dépendances de Jointure

Théorème de décomposition (Heath, Fagin)

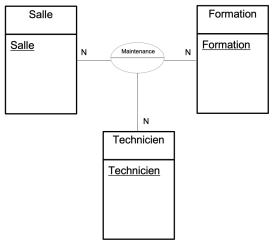
- Un schéma de relation R peut être décomposé sans perte de données en schémas de relation $R_1, ..., R_n$ si et seulement si la dépendance de jointure $\bowtie [R_1, ..., R_n]$ est valide sur R.

Remarque

- Sans perte de données : signifie que, quelque soit la relation r définie sur R, et ses projections $r_n = \pi_{R_n}(r)$ on a :

$$r = r_1 \bowtie ... \bowtie r_n$$

Exemple : Décomposition sans dépendance de jointure



Supposons la relation suivante, qui stocke une association ternaire entre des salles, des formations et des techniciens, sans aucune règle particulière.

Maintenance	SALLE (S)	FORMATION (F)	TECHNICIEN (T)
	C1	L3IF	SMITH
	C1	M2TIW	JAMES
	C1	L3IF	JAMES
	C2	L3IF	SCOTT

Essayons de décomposer en deux relations sur les projections : $r_1 = \pi_{S,F}(Maintenance)$ et $r_2 = \pi_{S,T}(Maintenance)$.

Exemple : Décomposition sans dépendance de jointure

r_1	SALLE (S)	FORMATION (F)
	C1	L3IF
	C1	M2TIW
	C2	L3IF

r_2	SALLE (S)	TECHNICIEN (T)
	C1	SMITH
	C1	JAMES
	C2	SCOTT

$r_1 \bowtie r_2$	SALLE (S)	FORMATION (F)	TECHNICIEN (T)
	C1	L3IF	SMITH
	C1	M2TIW	JAMES
	C1	M2TIW	SMITH
	C1	L3IF	JAMES
	C2	L3IF	SCOTT

Sur cet exemple, la jointure crée un tuple qui n'existait pas. La décomposition n'est pas garantie sans perte de données. On ne peut pas décomposer r.

Exemple : Décomposition en présence d'une DF

Maintenant, on suppose qu'une salle est dédiée à une seule formation : donc $S \rightarrow F$ doit être satisfaite dans toutes les relations. Ceci crée de la redondance, mais apporte aussi la solution...

Maintenance	SALLE (S)	FORMATION (F)	TECHNICIEN (T)
	C1	L3IF	SMITH
	C1	L3IF	JAMES
	C2	L3IF	SCOTT

Dans ce cas, la dépendance de jointure $\bowtie \{SF, ST\}$ est également satisfaite. On peut [6] décomposer sans risque de créer des tuples inexistants par jointure, en deux relations : $R_1(S, F)$ et $R_2(S, T)$.

6. et on doit !

Garantir une décomposition "sans pertes de données"

Comment être certain que la décomposition est sans perte de données ?

- OK si on a appliqué strictement le théorème de décomposition
  - On peut justifier chaque "décomposition" par une DF ou une DJ.
- Dans tous les cas : on peut tester une décomposition finale par une procédure de poursuite

Tester une décomposition par la poursuite - Exemple

Soit $R = ABCDE$, $\Sigma = \{E \rightarrow C; C \rightarrow E; \bowtie \{ABE, ABCD\}\}$. Les deux affirmations suivantes sont équivalentes :

- La décomposition de R en $\{ABD; ABCE\}$ est sans perte de données
- $\Sigma \models\!\!\bowtie \{ABD, ABCE\}$

Initialisation (les variables $x_A, x_B, ...$ sont remplacées par des entiers > 0) :

r	A	B	C	D	E
	0	0	1	0	1
	0	0	0	2	0

Tester une décomposition par la poursuite - Exemple

Soit $R = ABCDE$, $\Sigma = \{E \rightarrow C; C \rightarrow E; \bowtie \{ABE, ABCD\}\}$. Les deux affirmations suivantes sont équivalentes :

- La décomposition de R en $\{ABD; ABCE\}$ est sans perte de données
- $\Sigma \models\!\!\bowtie \{ABD, ABCE\}$

On "force" $\bowtie \{ABE, ABCD\}$:

r	A	B	C	D	E
	0	0	1	0	1
	0	0	0	2	0
	0	0	1	0	0
	0	0	0	2	1

Tester une décomposition par la poursuite - Exemple

Soit $R = ABCDE$, $\Sigma = \{E \rightarrow C; C \rightarrow E; \bowtie \{ABE, ABCD\}\}$. Les deux affirmations suivantes sont équivalentes :

- La décomposition de R en $\{ABD; ABCE\}$ est sans perte de données
- $\Sigma \models\!\!\bowtie \{ABD, ABCE\}$

On force $E \rightarrow C$

r	A	B	C	D	E
	0	0	0	0	1
	0	0	0	2	0
	0	0	0	0	0
	0	0	0	2	1

le troisième tuple est à 0, la décomposition est sans perte de données. On peut arrêter ici.

Décomposition sans perte de données

Théorème

Quelque soit la relation R et Σ un ensemble de contraintes, on peut toujours décomposer R est un ensemble de schémas qui sont tous en 5 FN.

Algorithme naïf

- Si une contrainte σ ne satisfait pas la 5FN pour un schéma R_i, décomposer R_i selon σ.
- Recommencer tant que tous les schémas de relation ne sont pas en 5FN.

Décomposition sans perte de données - Exemple

Soit $R = ABCDE$, $\Sigma = \{A \rightarrow BC; D \rightarrow E; \bowtie \{BE, ABCD\}\}$. R n'est pas en 5FN.

Procédons à des décompositions

1. Décomposition de R par $\bowtie \{BE, ABCD\}$: $R_1(BE), R_2(ABCD)$
   - R_1 est en 5FN (pas de dépendance)
   - R_2 est en 1FN à cause de $A \rightarrow BC$.
2. Décomposition de R_2 par $A \rightarrow BC$: $R_1(BE), R_2(ABC), R_3(AD)$
   - R_2 et R_3 sont en 5FN.

La décomposition est sans perte de données. Mais la DF $D \rightarrow E$ est perdue ! Elle ne peut pas être maintenue, et ne sera pas forcément satisfaite dans les jointures à venir.

Plan du chapitre

Notion de perte de dépendances

- On a vu comment décomposer une relation sans perte de données pour éviter la redondance
  - On pourra retrouver les tuples originaux par jointure.
- Mais ce n'est pas le seul objectif de la normalisation !
- Il faut pouvoir garantir la cohérence dans le temps
  - S'assurer que les DF seront respectées dans les MàJ futures
  - Chaque DF doit être exprimable dans une des relations
    - Ou impliquée par des DF exprimables (et donc être satisfaite dans la jointure)

Exemple

Une entreprise gère des projets, qu'elle affecte chacun à l'une de ses agences. Chaque employé est affecté à une agence. On considère la relation : $R(EMP, PROJ, AGENCE)$ munie de l'ensemble de dépendances $\Sigma = \{E \rightarrow A; P \rightarrow A\}$.

- Clé minimale : EP. La relation n'est pas en 3FN.
- Si on décompose selon $E \rightarrow A$:
  - On obtient $R_1(A, \underline{E})$ et $R_2(\underline{E, P})$. Les deux relations sont en 5FN.
  - La DF $P \rightarrow A$ est perdue
- La situation n'est pas meilleure si on décompose selon $P \rightarrow A$

Attention aux DF qui se cachent !

Chaque employé est affecté à un seul projet ; chaque projet est découpé en numéro de tâches (tâche 1, tâche 2, etc...) avec, pour chaque tâche, une date de fin prévue.
Soit $R = (Emp, Proj, Numtache, Fin)$ et
$\Sigma = \{E \to P; PN \to F; EN \to F\}$. La clé minimale est EN, la relation n'est pas en 3FN.

- En décomposant par $PN \to F$ puis par $E \to P$, on obtient :
  - $R_1(\underline{PN}F)$, $R_2(\underline{E}P)$, $R_3(\underline{EN})$
  - Cette décomposition est en 5FN.
- La DF $EN \to F$ n'est plus exprimable...
- ... mais peut-être déduite simplement des deux autres, qui elles s'expriment bien.

$EN \to F$ sera toujours satisfaite dans les jointures entre R_1, R_2 et R_3

Décomposition sans perte de dépendances - Définition

Soit R, Σ un schéma de relation muni d'un ensemble de dépendances, et soit $R_1, ..., R_n$ une décomposition de R. Soit $\Sigma' \subseteq \Sigma$ toutes les dépendances de Σ^+ [7] qui sont exprimables au sein des relations $R_1, ..., R_n$. On dit que la décomposition est sans perte de dépendance si Σ' est une couverture de Σ.

Pour tester la conservation des dépendances

La difficulté est bien sûr de calculer Σ'. On y trouve toutes les DF de Σ directement exprimables dans un des schémas $R_1, ..., R_n$, mais pas seulement.

7. $Sigma^+$ est l'ensemble de TOUTES les DF dérivables depuis Σ

Tester la perte de DF

Soit R, Σ un schéma de relation muni d'un ensemble de DF, et soit $R_1, ..., R_n$ une décomposition de R. Si les DF de Sigma ne sont pas trivialement exprimables dans les schémas (R_i), appliquer la méthode suivante.

1. Calculer une couverture minimale et réduite uniquement à gauche Σ_1 de Σ
2. Construire Σ_2 de la façon suivante :
   - Pour chaque $X \rightarrow Y \in \Sigma_1$ et pour $i = 1...n$
     - Si $X \subseteq R_i$ alors ajouter $X \rightarrow R_i \cap Y$ dans Σ_2
3. Vérifier que chaque partie gauche de Σ_1 conserve sa fermeture dans Σ_2

Perte de dépendances - Exemple 1

Soit $R = ABCDE$ un schéma de relation muni de $\Sigma = \{A \rightarrow BE; BC \rightarrow D; AC \rightarrow D\}$. Considérons la décomposition $\{ABE, BCD, AC\}$. Est-elle sans perte de DF ?
On fait l'inventaire des DF impliquées par Σ dans chaque schéma :

- $ABE : \{A \rightarrow BE\}$
- $BCD : \{BC \rightarrow D\}$
- $AC : \emptyset$

On retrouve déjà deux DF ; mais par ailleurs, $\{A \rightarrow BE; BC \rightarrow D\} \models AC \rightarrow D$.

Perte de dépendances - Exemple 2

Soit $R = ABCD$ un schéma de relation muni de $\Sigma = \{A \to B; B \to C; C \to D; D \to A\}$. Considérons la décomposition $\{AB, BC, CD\}$. La DF $D \to A$ serait-elle perdue ?
On fait l'inventaire des DF impliquées par Σ dans chaque schéma :

- $AB : \{A \to B; B \to A\}$
- $BC : \{B \to C; C \to B\}$
- $CD : \{C \to D; D \to C\}$

Si on calcule D^+ dans cet ensemble de DF, on trouve bien $D^+ = ABCD$.

Plan du chapitre

Algorithmes de normalisation

Triple objectif

- Garantir des relations normalisées
  - Pour évacuer les anomalies de mises à jour
- Sans perte de données
  - Pouvoir trouver des tuples cohérents, réels, par les jointure
- Sans perte de dépendances
  - Pouvoir assurer la cohérence des données dans les insertions futures

Deux approches

- Approche descendante
  - Décompositions successives basées sur les dépendances
- Approche ascendante
  - Construire une relation pour chaque dépendance

Normalisation par décomposition

Décomposer jusqu'à la 5FN

- Approche dite "descendante" car on part d'un schéma de relation R non normalisé.
- Intéressant pour corriger des anomalies simples
- Par exemple dans des normalisations *a posteriori*
  - Une ou deux dépendances qui créent de la redondance
- Garantie sans perte de données (théorème de décomposition)
- Atteint la 5FN dans tous les cas (on décompose tant que non atteinte)
- MAIS ne garantit pas la conservation des dépendances.

Forme générale d'un algorithme de décomposition

- On réduit les DF dans une couverture minimale avec réduction des parties gauches et droites
- Tant qu'il existe une relation R_i qui n'est pas en 5FN
  - Repérer une DJ $\bowtie \{R_{i_1}, ..., R_{i_n}\}$ qui contredit la 5FN
    - Utiliser les DF en priorité
    - Puis les DJ de taille 2
    - Puis enfin les DJ.
  - Remplacer alors R_i par les schémas $R_{i_1}, ..., R_{i_n}$.
- Conseil : Dans le cas de deux DF $X \rightarrow Y$ et $Y \rightarrow Z$, commencer par décomposer selon $Y \rightarrow Z$
  - La DF $X \rightarrow Y$ ne sera pas perdue.
- Remarque : au fur et à mesure de la décomposition, de nouvelles DJ peuvent apparaître...

Normalisation par synthèse

- Approche dite ascendante, chaque DF engendre un schéma de relation
- Plus intéressant dans le cas de problèmes complexes
- Méthode de conception, complémentaire E/A
  - Beaucoup de dépendances et d'attributs, interactions entre dépendances...
- Garanti sans perte de dépendance
- Garanti sans perte de données
- MAIS ne garanti que la 3ème Forme Normale
  - Lorsque la FNBC sans aucune perte est impossible

Forme générale d'un algorithme de synthèse

- 1) Construire Σ, une couverture minimale réduite des DF.
- 2) Générer une relation XY pour chaque DF $X \rightarrow Y \in \Sigma$;
- 3) On supprime les schémas de relation qui ne sont pas maximaux par inclusion.
- 4) S'il y a perte de données [8], alors on rajoute une relation R_{cle} composée d'une clé minimale dans Σ.
- 5) Vérifier alors la Forme Normale de chaque relation
  - 3FN : travail supplémentaire selon les cas
  - FNBC : étudier les DJ dans le cahier des charges pour décomposition éventuelle

8. Voir juste après

Focus sur l'étape 4

Comment savoir s'il y a perte de données à cette étape ? On est dans un cas particulier en raison de deux éléments :

- On ne travaille qu'avec des DF à ce stade (pas de DJ)
- Il est impossible d'avoir une perte de DF par l'approche de synthèse.

Dans ces conditions, les affirmations suivantes sont équivalentes :

- La décomposition est sans perte de données
- Un des schéma de relation est une clé de l'application

Remarques sur les DJ dans la normalisation

L'importance du schéma de relation

- Contrairement à une DF, une DJ fait référence au schéma de la relation :
  - $X \rightarrow Y$ est valide "globalement"
  - Dans l'expression $\bowtie [R_1, ..., R_2]$, il faut que $\cup R_i = R$
    - Exprime une indépendance, dépend de TOUS les attributs.
    - A considérer donc PENDANT la normalisation, en fonction des schémas obtenus, selon l'étude du cahier des charges.

Remarques sur les DJ dans la normalisation

Approches par décomposition

- Une fois décomposé en FNBC, étudier les DJ
- Pour garantir la 5FN

Approche par synthèse

- Pas de schéma de départ, bien souvent les DJ ne s'expriment pas encore.
- Dans les schémas en FNBC, étudier les DJ
- Puis appliquer la décomposition dans ces schémas si besoin
- Garantir ainsi la 5FN

Remarques sur les DJ dans la normalisation

Se concentrer sur certains schémas uniquement

- Toutes les clés minimales de taille 1 : 3FN $\Rightarrow$ 5FN
- Au moins une clé minimale de taille 1 : FNBC $\Rightarrow$ 5FN
  - Attention particulière à la relation R_{cle} ajoutée en fin d'algorithme de synthèse.
  - Elle regroupe ce que les DF n'ont pas "su" exprimer
    - Associations N-N sans attributs

Exemple

Considérons des recettes de cuisine. Chaque recette a un identifiant unique (A) et un nom (B), une description (C). Les recettes son composées d'une quantité donnée (H) de plusieurs ingrédients ; un ingrédient est caractérisé par son nom (D), il a également une description (E). Enfin, une recette utilise des ustensiles, qui ont un nom unique (F) et une description (G).

- 1) Inventaire des DF
  - $A \rightarrow BC$
  - $D \rightarrow E$
  - $AD \rightarrow H$
  - $F \rightarrow G$
- 2) On pourra vérifier que la couverture est bien minimale et réduite.
- 3) On a donc quatre relations (on les nomme relativement au contexte) :
  - $Recettes(\underline{A}BC)$
  - $Ingredeients(\underline{D}E)$
  - $Compose(\underline{AD}H)$
  - $Ustensiles(\underline{F}G)$

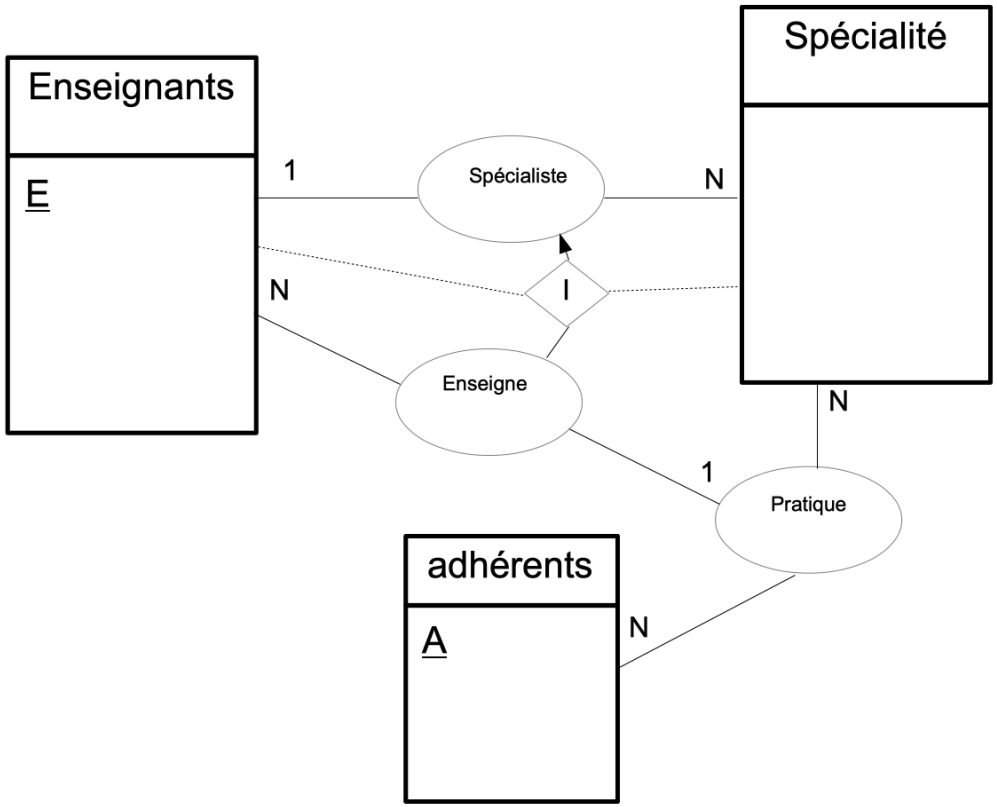
  Ces relations sont toutes en FNBC.

Exemple (suite)

- 4) On suspecte une perte de jointure, car la relation $ustensiles$ n'a aucun attribut commun avec le reste de la BD. En effet, la seule clé minimale de l'application est ADF, et n'est contenue dans aucun des schémas. On rajoute alors une relation $R_4(\underline{ADF})$. Ainsi, il n'y a plus de perte de jointure.
- 5) Dans cette relation, le cahier des charges induit la DJ $\bowtie [AD, AF]$ (une recette utilise un ensemble d'ingrédients indépendamment d'un ensemble d'ustensiles. Aucun lien direct n'est décrit entre les ingrédients et les ustensiles). Ainsi, R_4 n'est pas en $4FN$.
- 6) grâce à la DJ on décompose sans perte en deux relations $(\underline{AD})$ et $(\underline{AF})$. On supprime (AD) car elle est inclue dans la relation $Compose$. On donne le nom $Utilise$ à $(\underline{AF})$.
- La base de données est maintenant en 5FN.

Exemple de problème en 3FN

Dans un club de danse, des adhérents choisissent des spécialités ; on leur affecte un professeur pour chaque activité choisie, sachant que chaque professeur n'enseigne qu'une seule activité.



Exemple de problème en 3FN

Considérons la relation $Enseigne(AES)$. On a : $A, S \rightarrow E$ et $E \rightarrow S$. Il y a deux clés minimales : AS et AE ; la relation est en 3FN "à cause" de $E \rightarrow S$.

Décomposition sur $E \rightarrow S$?

On obtient bien deux schémas en FNBC $R_1(ES)$ et $R_2(AE)$, mais la DF $A, S \rightarrow E$ est perdue !

Algorithme de synthèse ?

Il conduit à créer les schémas de relation $R_1(AES)$ et $R_2(ES)$, puis à supprimer le deuxième en raison de l'inclusion ; donc on reste sur la relation (AES) en 3FN de départ...

Il n'existe pas de décomposition de ce problème en FNBC *sans perte de DF*.

Exemple de problème en 3FN

Que faire alors ?

La meilleure démarche consiste à :

- Conserver $R_1(AES)$ en déclarant bien la clé AS.
- conserver la relation $R_2(\underline{E}S)$, qui est en FNBC et permettra d'assurer la DF par la simple déclaration de la clé E
- Implanter une clé étrangère $R_1[ES] \subseteq R_2[ES]$
  - La partie droite est bien clé dans R_2 ; même si c'est une "superclé". On la déclarera UNIQUE de façon à ce que l'outil accepte la déclaration $FOREIGNKEY$.
  - Puisque tous les couples ES de R_1 seront issus de R_2, ce mécanisme assurera que la DF $E \rightarrow S$ soit bien toujours vraie dans R_1.

Ainsi, par le seul jeu des contraintes de base (clés et étrangère), la sémantique sera garantie. On aura presque toutes les qualités de la FNBC.

Plan du chapitre

Les valeurs NULL : le piège de la facilité

Deux significations pour une seule valeur

- La valeur est inconnue au moment de la saisie (et pour un certain temps...)
  - Acceptable dans les attributs de simple "description"
  - "Périlleux" dans les parties gauches des DF !
- La valeur ne s'applique pas.
  - C'est un défaut de conception à éviter au maximum dès le départ !
  - A considérer dans les choix dès le modèle E/A.

Les valeurs NULL seront une difficulé

- Pour analyser les données (data science)
- Pour normaliser à posteriori les relations
  - Compromet la notion de "perte de jointure".

Perte de jointure et valeurs NULL

Étudiants	NUMETUD	NOMETUD	NUMENS	NOMENS	DATEENCADRE
	28	Codd	5050	Tarjan	2015
	28	Codd	3434	Papadimitriou	2020
	32	Armstrong	2123	Mannila	2019
	53	Fagin	5050	Tarjan	2005
	107	Bunneman	NULL	NULL	NULL

Étudiants	NUMETUD	NOMETUD
	28	Codd
	32	Armstrong
	53	Fagin
	107	Bunneman

Énseignants	NUMENS	NOMENS
	5050	Tarjan
	3434	Papadimitriou
	2123	Mannila

Éncadre	NUMETUD	NUMENS	DATEENCADRE
	28	5050	2015
	28	3434	2020
	32	2123	2019
	53	5050	2005

La décomposition en 3 relations est correcte ; mais elle n'est pas "sans perte de jointure".

Plan du cours

Plan du chapitre

Améliorer les performances

La normalisation est un aspect logique

- On a travaillé sur le schéma relationnel, niveau logique
- Permet de travailler de façon déclarative
  - Niveau visible par les utilisateurs (développeurs)
  - Permet de formuler les requêtes SQL sur un modèle commun
- La performance est une autre question.
- Doit être considérée APRÈS la normalisation.
- Principalement au niveau physique du SGBD

Niveau physique

- Niveau d'abstraction qui intègre des éléments d'organisation, de méthodes d'accès...
- Attention, c'est différent du niveau matériel, ou du fonctionnement "interne" du SGBD (représentation des données sur le disque)

Le chemin d'une requête SQL

Plein de choses se passent après le " ;" !

- La requête est analysée (syntaxique, sémantique)
- La requête est comparée avec les requêtes récentes (exploitation du "cache")
- La requête est optimisée par le SGBD (cœur du métier !)
  - Choix d'un plan d'exécution : ordre des opérations.
  - Choix des méthodes d'accès
    - Quel index, quelle partition utiliser ?
  - Choix des algorithmes
    - En particulier les jointures : imbriquées ? fusion ? hachées ?
- Finalement, un programme machine est généré puis exécuté.

Plan du chapitre

Les vues

Une VUE est un alias de requête

- Dans les bases de données, une vue est une requête nommée.
  - Après création, s'utilise donc comme une relation.
  - Permet de simplifier l'écriture des requêtes
  - Permet de contrôler les accès aux utilisateurs
- En théorie, pas d'effet "direct" sur les performances
  - peut parfois "aider" l'optimiseur de requête...

On peut parfois "modifier" une vue (!)

- Les modifications s'appliquent dans les vraies données
- Mais la vue doit être très simple [a] (une seule relation dans le FROM, pas de group by, etc...)

a. Les possibilités dépendent des SGBD

Les vues matérialisées

Vues stockée comme une relation

- Matérialiser une VUE revient à "pré-calculer" une requête
- Cette fois, c'est une relation stockée physiquement comme une table
- Utilisées dans des requêtes, elles font gagner beaucoup de temps.
  - par exemple, pré-calculer une jointure qui est coûteuse et souvent utilisée.

Les vues matérialisées

La principale "question" est celle de la fraîcheur des données

- En fonction des spécifications des applications.
  - Peut-on tolérer des réponses peu fraîches aux requêtes ?
  - Si oui, dans quelle mesure ?
  - Cela doit être documenté dans le cahier des charges.
- à chaque modification des tables, ou de façon périodique [a]...

a. A lancer manuellement, ou automatiser, ou encore suivant les fonctionnalités du SGBD

Une vue matérialisée peut occuper également un espace important sur le disque.

Plan du chapitre

Partitionnement vertical

Principe

- Décomposer une relation "trop large" selon sa clé primaire
- Exemple :
  - $Ventes(\underline{numclient, numproduit, date}, facture, commentaires, etc...)$
  - On a très souvent besoin du triplet (numclient, numproduit, date)
  - Les autres champs sont plus rarement accédés, alourdissent les tuples
  - On crée $VentesPrincipale(\underline{numclient, numproduit, date})$
  - Et $VenteAnnexe(\underline{numclient, numproduit, date}, facture, commentaires, etc...)$
  - Pas de limite dans le nombre du moment qu'on conserve partout la clé primaire.
- Aucune dégradation de la forme normale.

Partitionnement vertical

Inconvénients

- Surcharge pour le concepteur, non géré par le SGBD
- Il faut changer le schéma, assurer l'intégrité
  - Suppressions, modifications dans la clé...

Utiliser des vues matérialisées

- Créer les décompositions comme des vues matérialisées
- Sans toucher à la relation d'origine
- Plus sécurisant, plus facile

Partitionnement horizontal

Principe général

- Séparer les tuples dans plusieurs tables selon une clé de partitionnement
  - Deux tuples dans des partitions différentes ont des valeurs différentes sur cette clé.
- Beaucoup plus utilisé que le partitionnement vertical
- Assisté par les SGBD [a] : le partitionnement est alors déclaratif.
  - selon une clé de partitionnement (une ou plusieurs colonnes)
  - Chaque tuple est dans une seule partition
  - La table doit être déclarée "partitionnée" à la création [b]
  - On peut supprimer ou ajouter des partitions par la suite.

---

a. Capacités variables selon les SGBD

b. Seules les partitions seront en réalité stockées, mais c'est transparent.

Partitionnement horizontal

Différentes méthodes

- A la déclaration, on décide de la méthode de partitionnement
  - Par intervalles sur la clé de partitionnement
    - Intervalles de date, alphabétiques, numériques
  - Par valeurs
    - Chaque partition porte sur une liste de valeurs de la clés de partitionnement
  - Par hachage
    - Répartition en N partitions
    - Transformation de la clé de partitionnement en nombre
    - Division euclidienne par N-1
    - le reste, compris entre 0 et N-1, indique la partition de destination du tuple.

Partitionnement horizontal

Exploitation

- La requête porte sur la table "complète"
  - Partitionnement invisible pour les développeurs
  - L'optimiseur choisi les partitions
- Intéressant si clé de recherche dans clause WHERE
- Très efficace pour supprimer toute une partition
- Pratique pour séparer les données le plus anciennes, moins accédées
- Il est conseillé en général de créer un index sur la clé de partitionnement

PostgreSQL : toutes les contraintes d'unicité doivent contenir la clé de partitionnement (car il n'y a pas d'index global)

Plan du chapitre

Généralités

Chaque tuple a un identifiant interne dans la base

- Un index fourni des identifiants [a] des tuples d'une table
- En fonction d'une clé de recherche [b]
- Dont les valeur sont utilisées dans les clauses WHERE (ou JOIN)
- Évite des parcours séquentiels de tables
- C'est le principal outil pour la performance
  - A privilégier s'il résout les problèmes
- L'utilisation est déclarative, les algorithmes transparents

---

a. Qu'on assimilera à des pointeurs

b. Qui n'est pas forcément la clé de la relation ; ces concepts sont sans lien entre eux.

Généralités

Principales structures d'index

- B-arbre
- Bitmap
- Table de hachage

Le choix est guidé par le besoin

- Sélectivité des attributs à indexer
- Type de requêtes (= , <= , >=, ...>)
- Capacités du SGBD [a]

a. Postgres v16 supporte les trois types (et d'autres encore)

Principes de base

Organisation en blocs

- la mémoire est organisée en "blocs" ou "pages"
  - Tous les tuples sont dans des blocs sur le disque.
  - Unité d'E/S : on ne charge pas "un tuple", mais un bloc.
  - Taille fixe pour la BD, décidé à l'installation : 1k, 2k, 4k, 8k , 16k...
  - Relation sur le disque = chaînage de blocs
- Les index sont également organisés en blocs chaînés

Principe de base

Index séquentiel, ou index plat

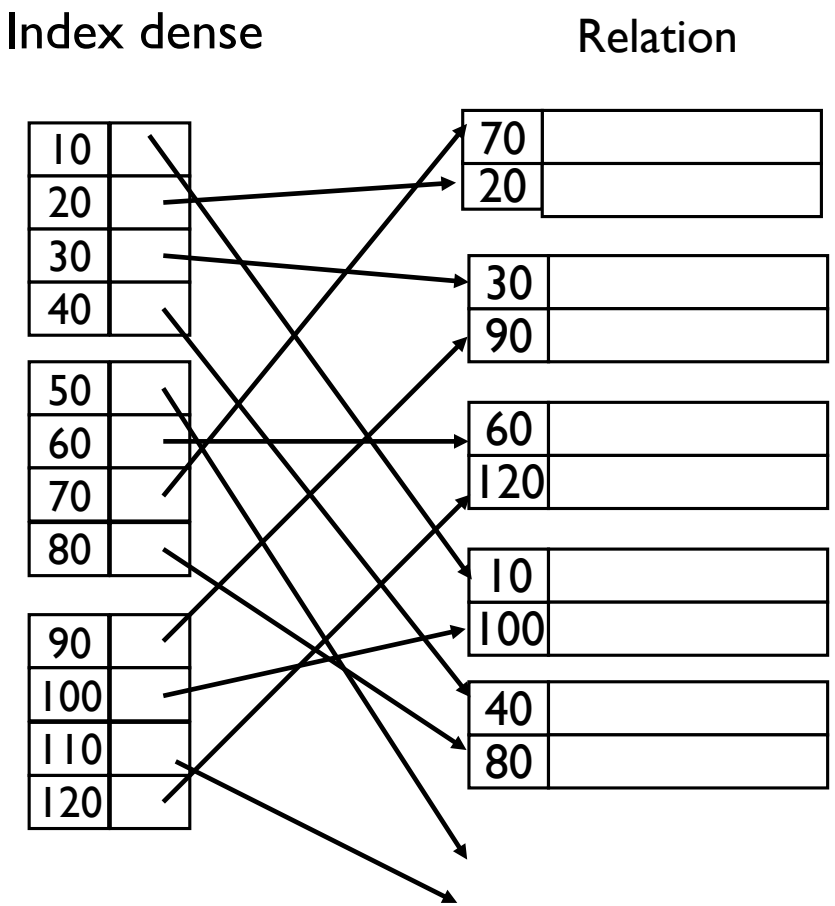
- Un index séquentiel est une simple liste de paires (clé, pointeur)
  - Maintenu trié sur les valeurs de clé
  - Donc recherche dichotomique en $\mathcal{O}(log(n))$
- Il peut être :
  - Dense : une paire (clé, pointeur) pour chaque valeur de clé
  - Creux : une paire (clé, pointeur) pour chaque bloc de la relation
    - Si les données de la relation sont triées.

Index Dense

Le fichier index requiert beaucoup **moins** de blocs que le fichier de données.
☞ plus facile à

charger en mémoire

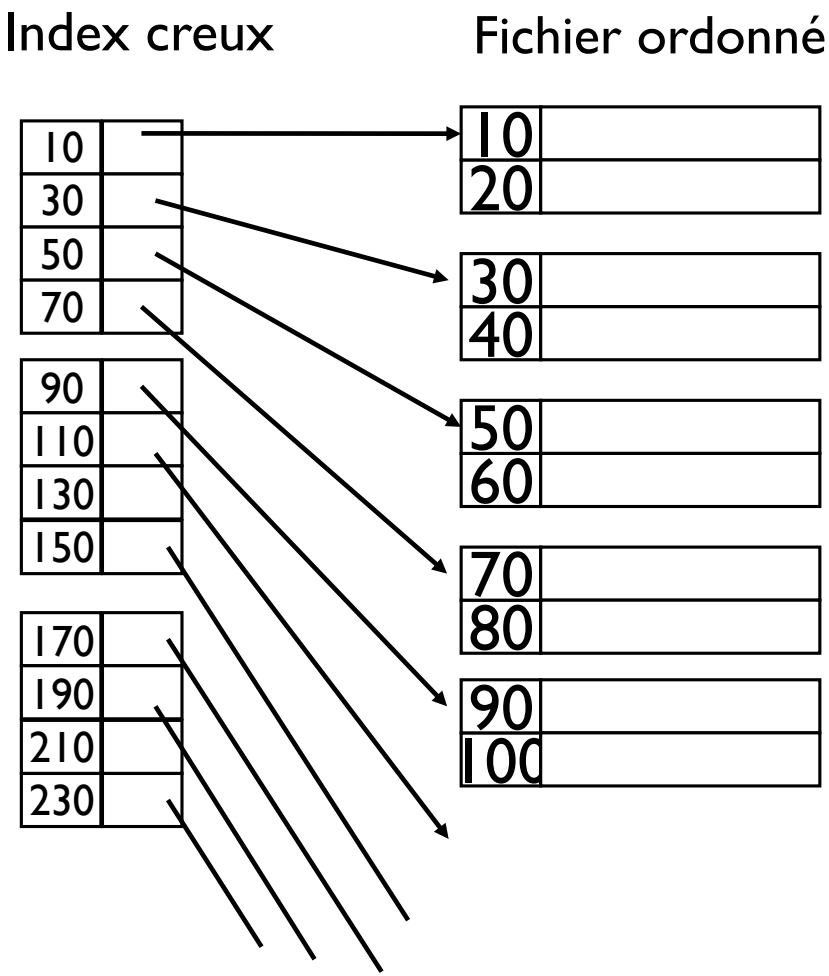
Pour une clé K, seulement $\mathbf{log_2 n}$, blocs d'index doivent être parcourus.



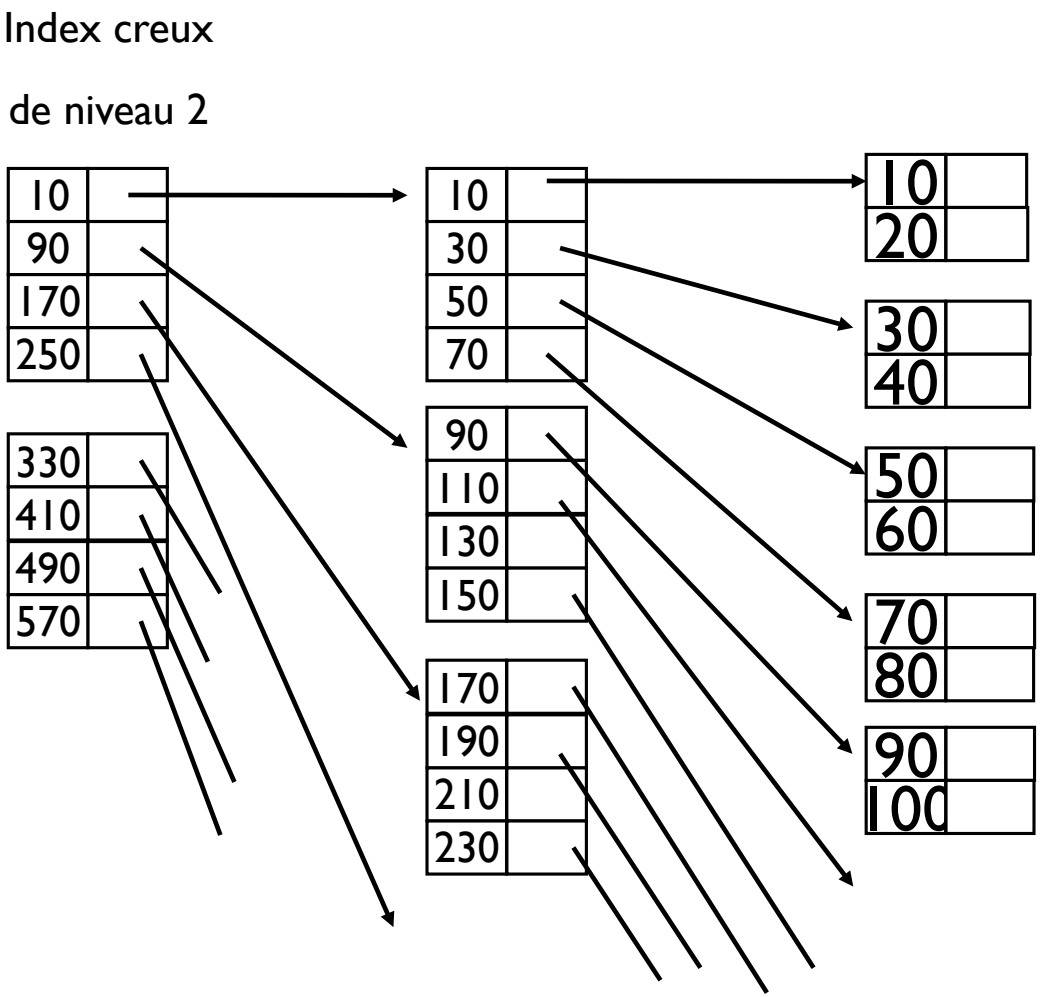
Index Creux

Une seule clé par bloc de données

Trouver l'entrée avec la plus grande valeur inférieure ou égale à la valeur recherchée



Index à plusieurs niveaux



Arbre équilibré (B-Arbre)

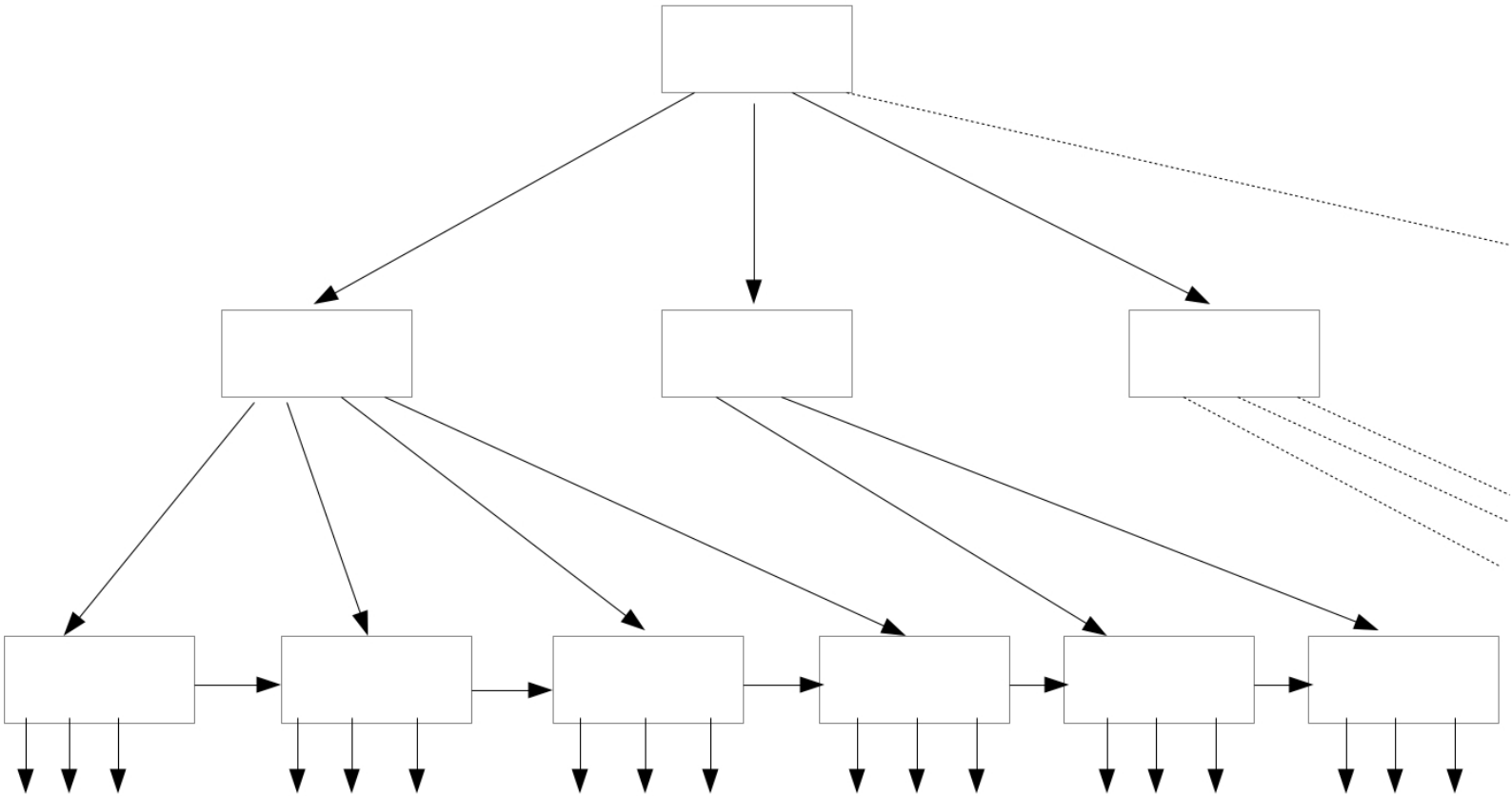
Généralisation des niveaux par un B-Arbre

- B-Arbre = Arbre équilibré (Balanced Tree)
- Tous les chemins (Racine –> Feuille) sont de même taille
- Permet une complexité constante de chaque recherche
- Structure la plus utilisée par les SGBD
  - Automatique pour les clés primaires et contraintes uniques
  - Par défaut dans les commandes "CREATE INDEX"

Description

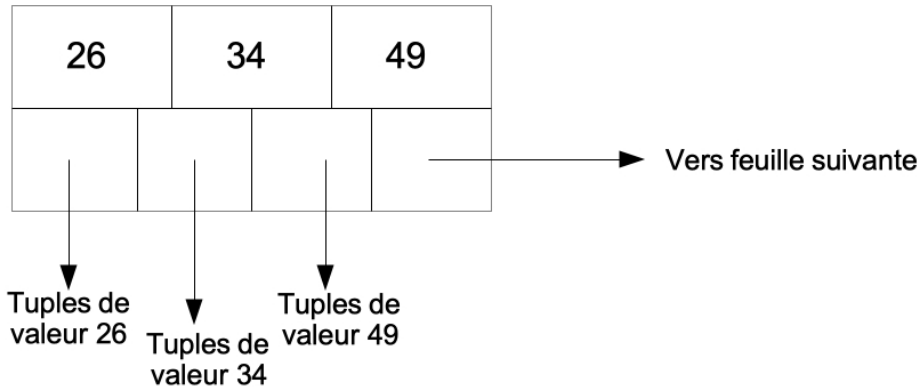
- Le niveau feuille est un index séquentiel
- Chaque élément de l'arbre (noeud ou feuille) est un bloc
- Chaque bloc contient N valeurs de clé et $N + 1$ pointeurs
- N dépend de la taille de codage de la clé et la taille des blocs

B-Arbre - structure générale

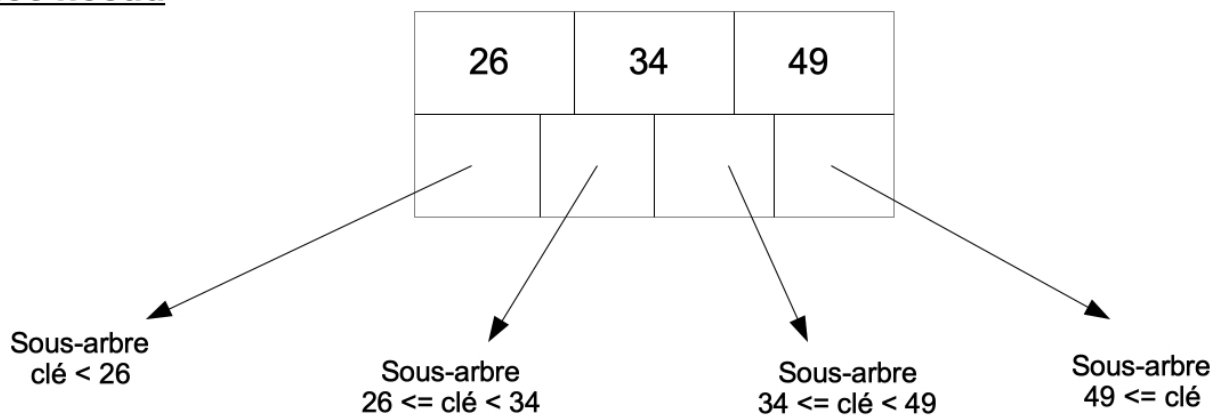


B-Arbre - détail des noeuds

Bloc feuille



Bloc noeud



Taille et complexité du B-Arbre

Exemple : B-arbre sur un attribut INTEGER (4 octets)

- Supposons 10 millions de valeurs distinctes dans la relation
- Taille des blocs de la BD : 8k (8192 octets)
- Taille des pointeurs (identifiants de tuples TID) : 4 octets
- Remplissage des blocs à 80% [a]

a. Permet plus de souplesse pour les mises à jour

Taille et complexité du B-Arbre

Nombre N de clés par bloc ?

- N valeurs et $N+1$ TID par bloc
- $4N + 4(N+1) \leq \frac{80}{100} \times 8192$ ===> $N <= 1023,5$
- $N = 1023$ valeurs de clé par bloc

Nombre de feuilles dans le B-Arbre ?

- $\lceil \frac{10000000}{1023} \rceil = 9776$ blocs feuille
- Parcours dichotomique des feuilles $\lceil Log(9776) \rceil = 14$ E/S

Coût d'une recherche dans l'arbre ?

- Taille d'un chemin (racine-feuille) : $\lceil Log_{N+1}(10000000) \rceil = 3$
- 3 E/S (mais la racine reste souvent en cache)
- Ajouter 1 E/S pour le chargement du bloc cible de la relation

Table de hachage

Hacher la clé par une fonction

- La fonction f retourne un nombre compris entre 0 et $N-1$
  - N est donc le nombre de "groupes" qui sera créé
- $f(cle)$ indique le groupe (bloc) à explorer
- Une bonne fonction de hachage :
  - est évaluée très rapidement
  - répartit de façon homogène les valeurs
  - Fonction classique : $f(X) = X\ MOD\ N$ (crée n groupes)

Table de hachage

Comparaison avec les B-Arbres

- Hachage plus rapide en accès direct
- Hachage construit rapidement sur des données statiques
- Hachage prend peu de place en mémoire
- B-Arbre permet les requêtes d'intervales
- B-Arbre a un meilleur comportement dans les maj

Utilisation dynamique très fréquente

- Le hachage est très utilisé "à la volée" par le SGBD
- Index temporaire d'une table, le temps d'une exécution
- Très fréquent lors de l'évaluation des jointures.

Principe de l'index Bitmap

Une matrice de 0 et 1

- Une ligne i par valeur de la clé d'indexation
- Une colonne j par tuples de la relation
- La case vaut 1 si le tuple j possède la valeur i, 0 sinon

Index bitmap - exemple

Exemple : codification de films

rang	titre	genre	...
1	Vertigo	Suspense	...
2	Brazil	Science-Fiction	...
3	Twin Peaks	Fantastique	...
4	Underground	Drame	...
5	Easy Rider	Drame	...
6	Psychose	Drame	...
7	Greystoke	Aventures	...
8	Shining	Fantastique	...
...	...	...	...

Index bitmap - exemple

Exemple

	1	2	3	4	5	6	7	8
Drame	0	0	0	1	1	1	0	0
Science-Fiction	0	1	0	0	0	0	0	0
Comédie	0	0	0	0	0	0	0	0

	9	10	11	12	13	14	15	16
Drame	0	0	0	0	0	0	1	0
Science-Fiction	0	1	1	0	0	0	0	0
Comédie	1	0	0	1	0	0	0	1

Index bitmap - Intérêt

- Intéressant si faible sélectivité (peu de valeurs de clé)
- Très efficace pour les combinaisons de contraintes
  - ET, OU entre bitmaps.
- Ne prend pas en compte les requêtes d'intervalles.
- Très utilisé en interne par les SGBD durant l'exécution
  - Par exemple pour construire un vecteur de correspondance de tuples après la lecture d'un index.
  - Puis s'en aider pour parcourir la relation en visitant uniquement les bons blocs.

Compléments sur les index

- La clé peut porter sur une séquence d'attributs
  - L'ordre des attributs est alors primordial
  - Un index $I < A_1, A_2, A_3 >$ sera utilisé :
    - Pour les requêtes sur A_1, (A_1, A_2) et (A_1, A_2, A_3)
    - Mais pas pour les requêtes sur A_2, A_3, A_2, A_3
- On peut indexer une fonction sur un attribut.
  - " [...] WHERE upcase(name)='SMITH' [...]"
  - Un index sur "name" ne sera pas utilisé !
  - Créer un index sur "upcase(name)"

Compléments sur les index

Les index ont un coût ! ! !

- Ne pas indexer n'importe quelle colonnes
- Coût en espace, coût des mises à jour
- Difficulté pour l'optimiseur (augmente les possibilités)
- Toujours guidé par le besoin
  - pour les requêtes trop lentes
  - Les B-Arbre permettent les requêtes d'intervalle
  - Pas les tables de hachage ni les bitmap
- On peut faire des index partiels
  - A partir d'une condition d'inclusion ou d'exclusion des tuples
  - Permet de ne pas indexer des valeurs peu recherchées.

Plan du chapitre

L'optimisation de requête

Un problème exponentiel

- L'espace de recherche est exponentiel
- Pas d'algorithme polynomial
- Un choix à faire en moins d'une seconde !
- Résultat exact impossible à partir de 7 à 10 relations dans le FROM...
  - Car il faut déterminer l'ordre des jointures
- L'optimiseur utilise alors des heuristiques
- On peut paramétrer des objectifs d'optimisation
  - Ex. : réponse exhaustive ou les premiers tuples ?

L'optimiseur utilise des statistiques sur les données, qu'il faut bien mettre à jour après des changements dans la base.

Ce qui influence l'optimisation

La façon d'écrire les requêtes a une influence !

- L'ordre dans lequel on écrit les jointures (JOIN)
- Ne pas utiliser DISTINCT s'il est inutile
  - L'optimiseur ne détectera pas l'inutilité et fera un tri pour rien
- Utiliser UNION ALL au lieu de UNION
- Bien choisir le connecteur : IN, NOT IN, EXISTS, NOT EXISTS
- Eviter au maximum les disjonctions (OR)

Les types de jointures

L'optimiseur doit choisir comment faire la jointure

- Jointure en boucle imbriquée
  - complexité $\mathcal{O}(N \times M)$
  - intéressant pour des très petites relations
  - ne nécessite quasiment pas de RAM (contexte embarqué)
- Jointure en tri fusion
  - Tri de chaque opérande, puis fusion
  - $\mathcal{O}(N(Log(N)) + M(Log(M)) + TailleResultat)$
- Jointure par hachage
  - Hachage en mémoire de la table la plus petite
  - Accès par hachage pour chaque tuple de l'autre table
  - $\mathcal{O}(c(N + M) + TailleResultat)$, c coût constant de la fonction de hachage

Remarque : dans le pire cas, $TailleResultat = N \times M$ (si tous les tuples se combinent, ce qui est hautement artificiel !).

Ainsi, "plus" le nombre de tuples participent à la jointure des deux côtés, "plus" la jointure imbriquée se justifie.

Visualiser les choix de l'optimiseur

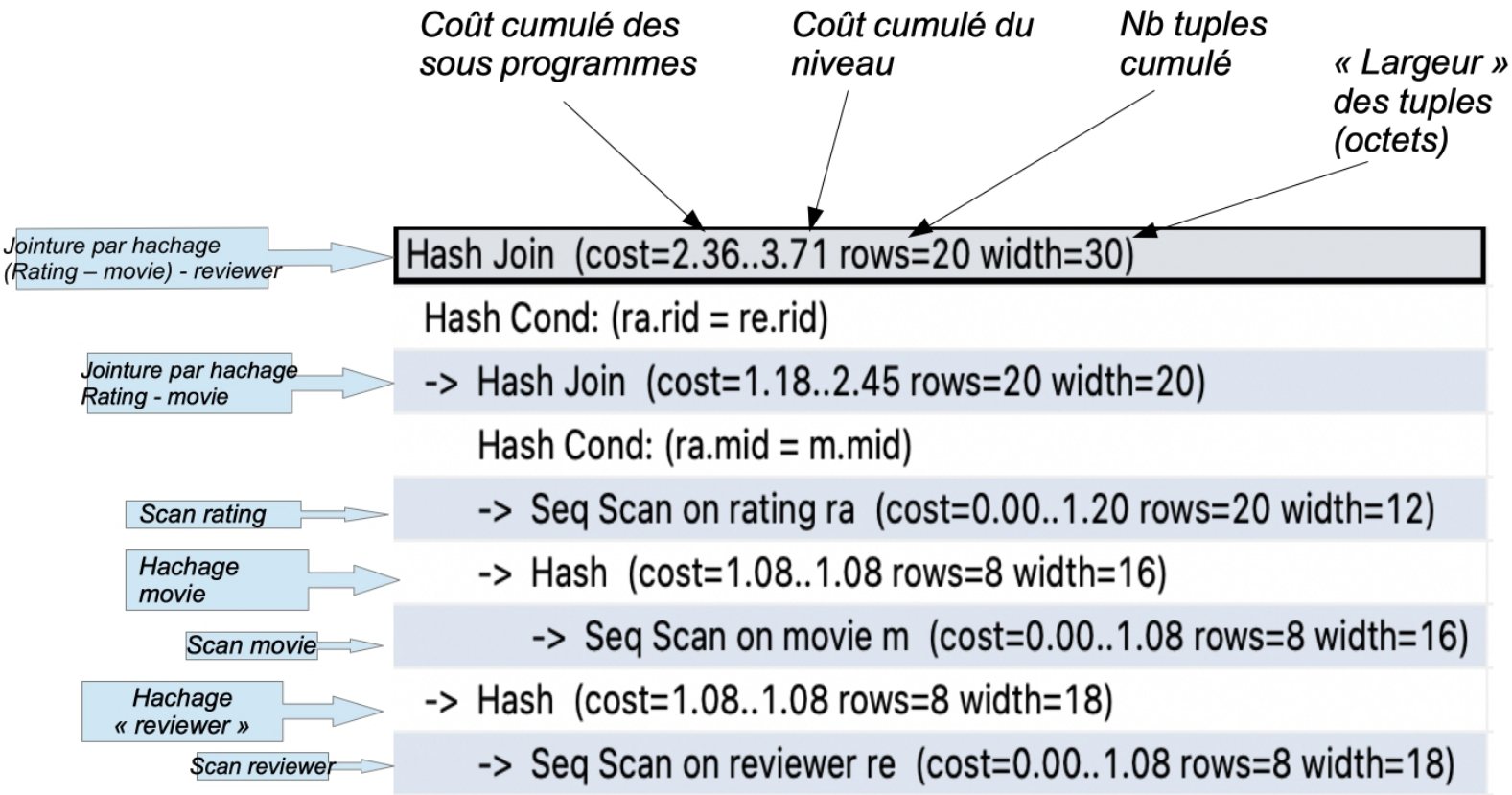
En cas de requêtes trop lente

- On visualise le plan choisi par l'optimiseur
- Sous PostgreSQL : "EXPLAIN" devant la requête.
- Il indiquera alors sous une forme d'arbre
  - L'ensemble des étapes et leur ordre
  - Les index utilisés
  - Les algorithmes de jointure choisis
  - Le coût estimé de chaque étape [a]
- "EXPLAIN ANALYSE" : fournit, en plus, le temps réel.
  - Peut varier d'une exécution à l'autre (cache, réseau, occupation du serveur...)

a. Selon des modèles de coût paramétrables

Exemple "EXPLAIN" de Postgres

```
EXPLAIN
SELECT m.title, re."name", ra.stars
FROM rating ra  JOIN movie m using(mid)
                JOIN reviewer re using(rid);
```



Exemple "EXPLAIN" de Postgres

```
EXPLAIN
SELECT m.title, re."name", ra.stars
FROM rating ra  JOIN movie m using(mid)
                JOIN reviewer re using(rid)
WHERE stars<2;
```

```
Hash Join  (cost=2.40..3.52 rows=1 width=30)
  Hash Cond: (re.rid = ra.rid)
  -> Seq Scan on reviewer re  (cost=0.00..1.08 rows=8 width=18)
  -> Hash  (cost=2.38..2.38 rows=1 width=20)
       -> Hash Join  (cost=1.26..2.38 rows=1 width=20)
            Hash Cond: (m.mid = ra.mid)
            -> Seq Scan on movie m  (cost=0.00..1.08 rows=8 width=16)
            -> Hash  (cost=1.25..1.25 rows=1 width=12)
                 -> Seq Scan on rating ra  (cost=0.00..1.25 rows=1 width=12)
                      Filter: (stars < 2)
```