Bases de données

Jacques Le Maitre

Université du Sud Toulon-Var

Bibliographie

- C. J. Date, *Introduction aux bases de données*, Vuibert, 2001.
- P. Delmal, *SQL2 - SQL3 Applications à ORACLE*, De Boeck Université, 2001.
- H. Garcia-Molina, J. D. Ullman, J. Widom, *Database Systems: the Complete Book*, Computer Science Press, 2002.
- G. Gardarin, *Bases de données*, Eyrolles, 2003.
- P. M. Lewis, A. Bernstein, M. Kifer, *Databases and Transaction Processing. An Application-Oriented Approach*, Addison-Wesley, 2002.
- J. Melton, A. R. Simon, *SQL:1999 Understanding Relational Language Components*, Morgan Kaufman Publishers, 2002.

Table des matières

Introduction

Qu'est ce qu'une base de données ?

- Une **base de données** (BD) est un ensemble d'informations archivées dans des mémoires accessibles à des ordinateurs en vue de permettre le traitement des diverses applications prévues pour elles.
- L'intérêt d'une BD est de regrouper les données communes à une application dans le but :
  - d'éviter les redondances et les incohérences qu'entraînerait fatalement une approche où les données seraient réparties dans différents fichiers sans connexions entre eux,
  - d'offrir des langages de haut niveau pour la définition et la manipulation des données,
  - de partager les données entre plusieurs utilisateurs,
  - de contrôler l'intégrité, la sécurité et la confidentialité des données,
  - d'assurer l'indépendance entre les données et les traitements.
- Les bases de données sont gérées par des logiciels spécialisés appelés **systèmes de gestion de bases de données** (SGBD).

Le modèle ANSI-SPARC

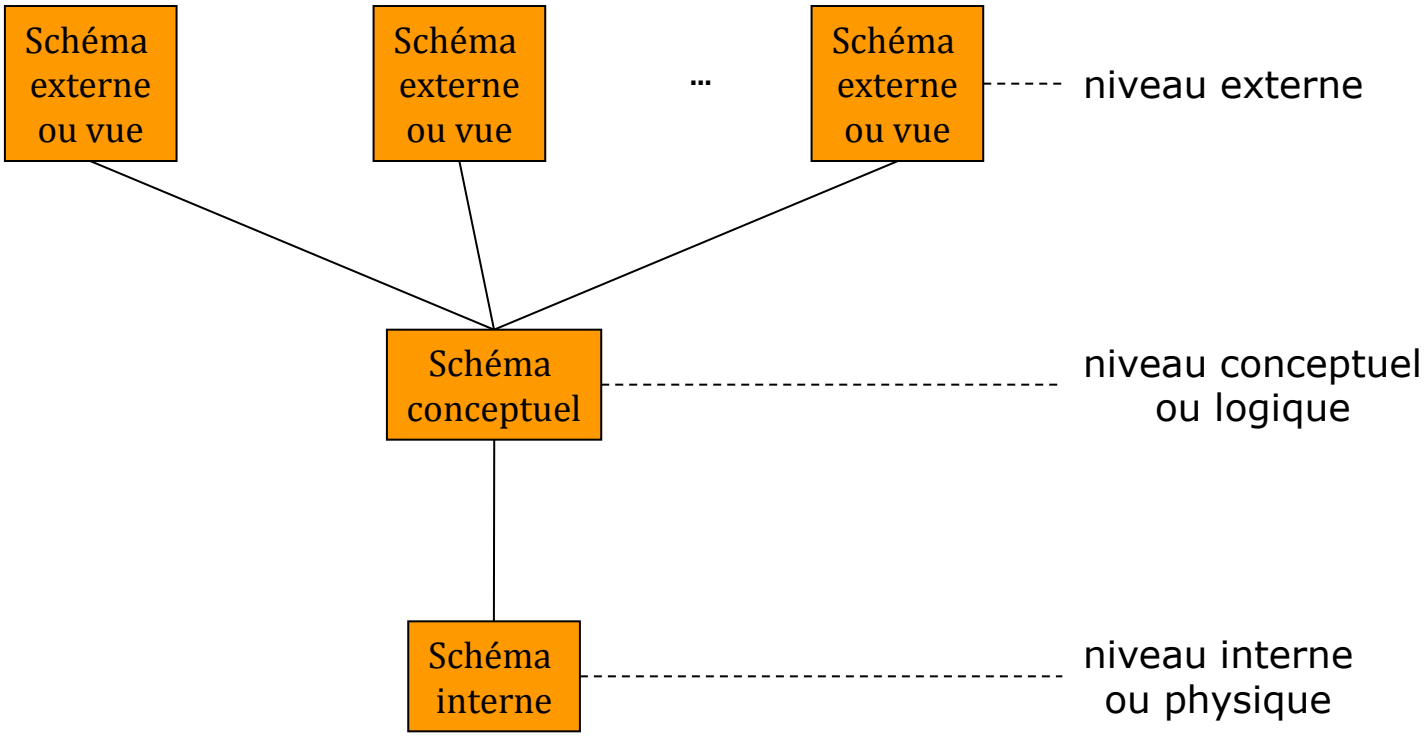


Schéma conceptuel

- Le schéma conceptuel est une représentation du monde réel auquel se rapporte la BD.
- Principaux concepts :
  - **entité** (ou **objet**) : une personne, un livre
  - **propriété** (ou **attribut**) : le titre d'un livre, l'adresse d'une personne
  - **association** : entre une personne et le livre dont elle est l'auteur
  - **agrégat** : une adresse composée d'une rue et d'un code postal
  - **collection** : un ensemble de personnes, une liste de prénoms

Principaux modèles conceptuels

- 1ère génération :
  - hiérarchique (IMS d'IBM)
  - réseau (DBTG CODASYL)
- 2e génération :
  - relationnel
  - entité-association
- 3e génération :
  - modèle orienté objet
  - modèle objet-relationnel
  - UML
- 4e génération (les données du Web)
  - XML

Exemple

- Soit une BD décrivant les livres d'un bibliothèque et leurs auteurs.
- On suppose qu'un livre est identifié par sa cote et un auteur par son nom.
- On se place à l'instant où la bibliothèque contient :
  - un seul livre,
  - ayant la cote BD/46 et le titre « Les BD en BD »,
  - écrit par Jean Dupont né en 1960 et Pierre Durand né en 1953.

Exemple en relationnel

- Schéma
  - `livre(cote: texte, titre: texte)`
    `auteur(nom: texte,`
    `prénom: texte,`
    `année_naissance: entier)`
    `écrire(cote: texte, nom: texte)`
- Extension

livre	
cote	**titre**
BD/46	Les BD en BD

auteur		
nom	**prénom**	**année_naissance**
Dupont	Jean	1960
Durand	Pierre	1953

écrire	
cote	**nom**
BD/46	Dupont
BD/46	Durand

Exemple en orienté objet

- Schéma
  - ```
    classe livre
      attribut cote: texte
      attribut titre: texte
      attribut écrit_par: liste(auteur)
    classe auteur
      attribut nom: texte
      attribut prénom: texte
      attribut année_naissance: entier
      méthode age(): entier {année_courante - self->année_naissance}
    livres: ensemble(livre)
    auteurs: ensemble(auteur)
    ```
- Instances
  - (l1, {cote = "BD/46", titre = "Les BD en BD", écrit_par = [a1,a2]})
    (a1, {nom = "Dupont", prénom = "Jean", année_naissance = 1960})
    (a2, {nom = "Durand", prénom = "Pierre", année_naissance = 1953})
- Variables persistantes
  - `livres = {l1}`
    `auteurs = {a1, a2}`
    ```

Exemple en XML (1)

```
<!ELEMENT bd (livres, personnes)
<!ELEMENT livres (livre*)>
<!ELEMENT livre (cote, titre, écrit_par*)>
<!ELEMENT cote (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT écrit_par EMPTY>
<!ATTLIST écrit_par ref IDREF>
<!ELEMENT auteurs (personne*)>
<!ELEMENT auteur (nom, prénom, année_naissance)>
<!ATTLIST auteur id ID>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT année_naissance (#PCDATA)>
```

Exemple en XML (2)

```
<bd>
 <livres>
  <livre>
   <cote>BD/46</cote>
   <titre>Les BD en BD</titre>
   <écrit_par ref="a1"/><écrit_par ref="a2"/>
  </livre>
 </livres>
 <auteurs>
  <auteur id="a1">
   <nom>Dupont</nom>
   <prénom>Jean</prénom>
   <année_naissance>1960</année_naissance>
  </auteur>
  <auteur id="a2">
   <nom>Durand</nom>
   <prénom>Pierre</prénom>
   <année_naissance>1953</année_naissance>
  </auteur>
 </auteurs>
</bd>
```

Schéma externe

- Un **schéma externe** représente la façon dont un utilisateur final ou un programme d'application voit la partie de la BD qui le concerne.
- Il existe en général plusieurs modèles externes pour une même BD.
- Le schéma conceptuel d'une BD peut être complexe : les schémas externes donnent aux utilisateurs une vision plus simple de ce schéma.
- Les schémas externes permettent aussi de protéger la BD contre des manipulations incorrectes ou non autorisées, en cachant certaines données à certains utilisateurs.

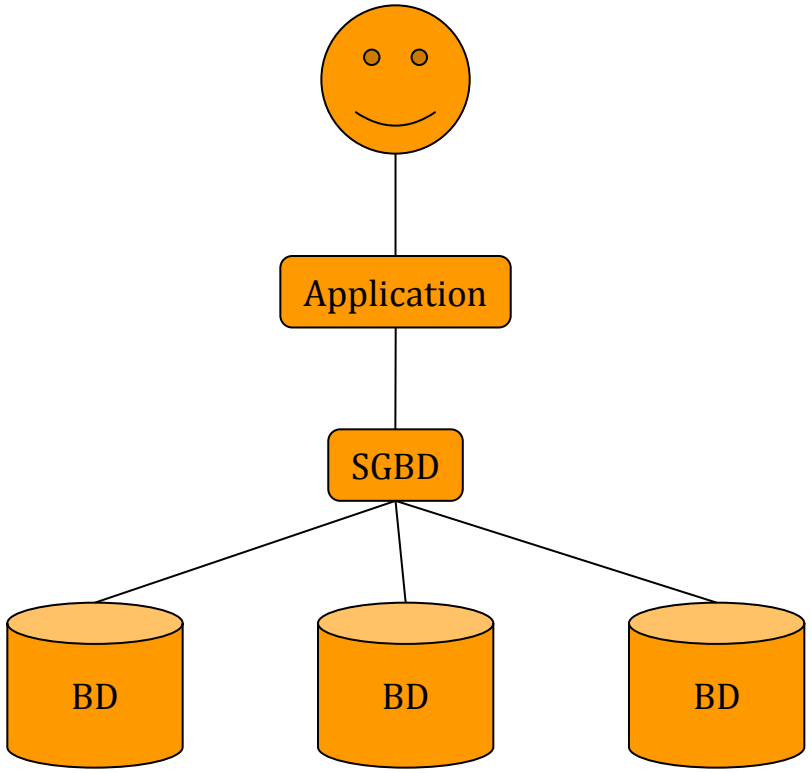
Schéma interne

- Le **schéma interne** décrit l'organisation des données en mémoire secondaire (sur disque) et la façon d'y accéder.
- L'organisation choisie doit permettre :
  - d'accéder le plus rapidement possible à un ensemble de données vérifiant certaines conditions,
  - de créer, modifier ou supprimer des données avec une réorganisation minimale et une utilisation optimale de la place disponible.

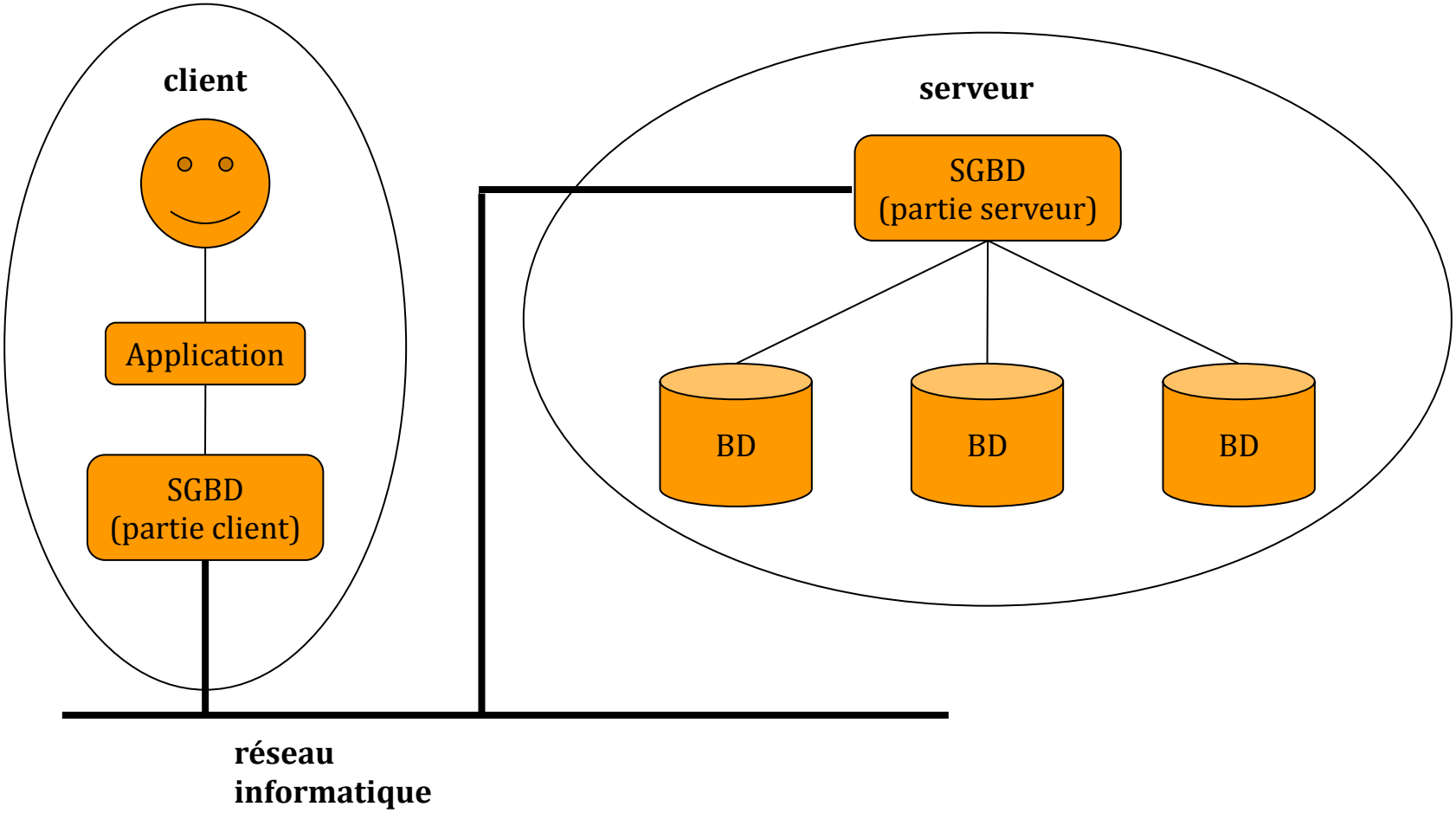
Architecture d'un SGBD

- On distingue 3 grands types d'architecture :
  - architecture centralisée,
  - architecture client-serveur,
  - architecture trois-tiers.

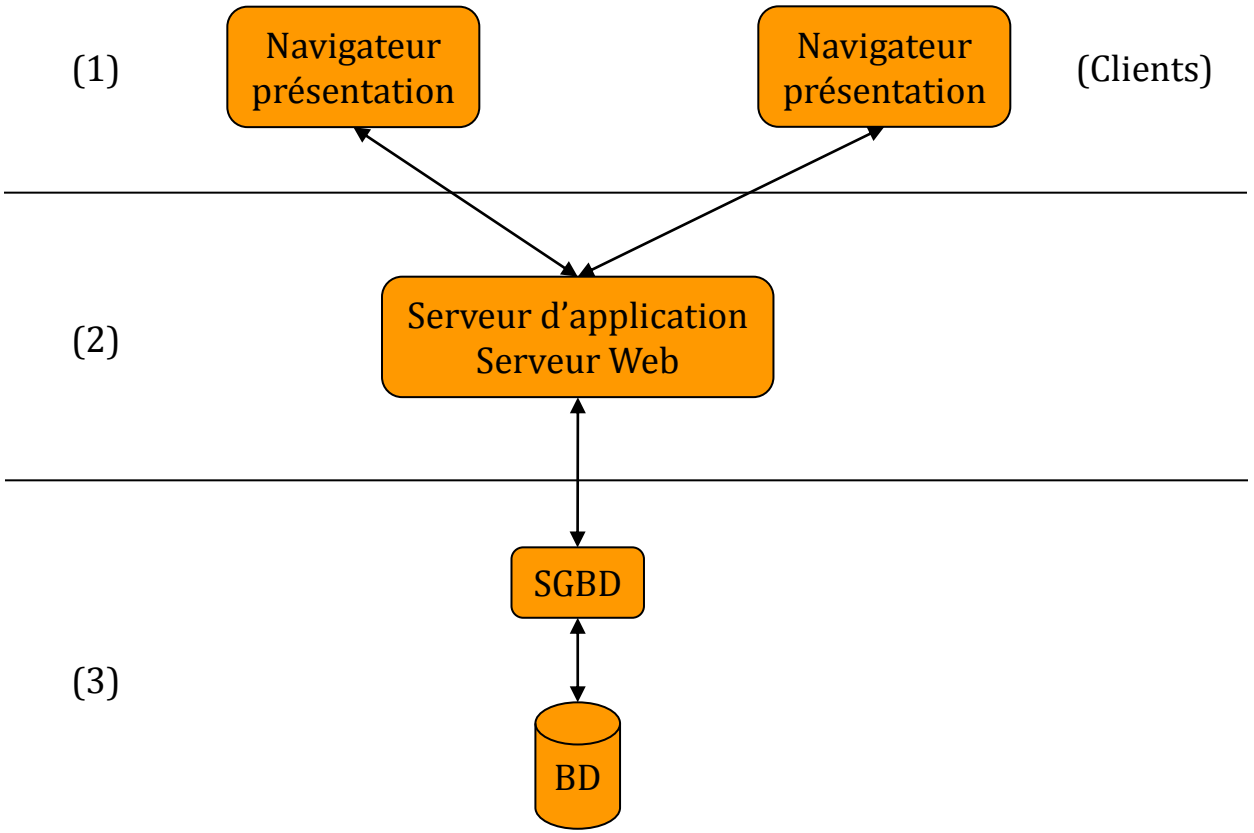
Architecture centralisée



Architecture client-serveur



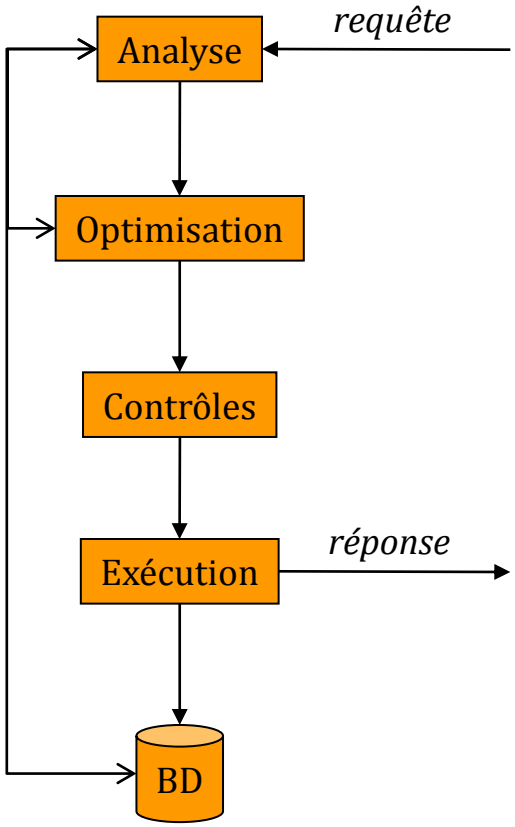
Architecture trois tiers



Langage de bases de données

- C'est au travers d'un langage déclaratif sont réalisées la définition et la manipulation d'une BD.
- Le plus connu et le plus utilisé de ces langages est **SQL.**
- Modes d'utilisation :
  - **intégré** dans un langage hôte (C, Java…),
  - **autonome**.

Exécution d'une requête



Contrôles

- **Intégrité**
  - Les données stockées dans une BD doivent respecter un certain nombre de contraintes dites d'intégrité. Un SGBD doit assurer qu'elles sont toujours respectées.
- **Confidentialité**
  - Un SGBD doit permettre d'interdire à certaines personnes de réaliser certaines opérations sur une partie ou sur toute la BD.
- **Concurrence**
  - Plusieurs utilisateurs se partagent la même BD. Un SGBD doit gérer les conflits qui peuvent se produire lorsqu'ils manipulent simultanément les mêmes données de façon à ce que la BD ne soit pas mise dans un état incohérent.
- **Reprise après panne**
  - Après une panne, qu'elle soit d'origine logicielle ou matérielle, un SGBD doit être capable de restaurer la BD dans un état cohérent, le même ou le plus proche de celui précédant la panne.

Transactions

- Pour un SGBD l'unité de traitement est la **transaction**.
- Une transaction est un fragment de programme qui fait passer une BD d'un état cohérent à un autre état cohérent, par une suite d'actions élémentaires.
- Un exemple classique de transaction est l'opération qui transfère une somme *S* d'un compte bancaire *A* à un compte bancaire *B* :

```
début transaction
  solde(A) = solde(A) - S
  solde(B) = solde(B) + S
fin transaction
```

- Il est clair que cette opération ne doit pas être interrompue entre le débit de *A* et le crédit de *B*.

Indépendance données-traitements

- **L'indépendance données-traitements** est indispensable pour pouvoir faire évoluer facilement l'organisation logique ou physique d'une BD ou bien l'architecture matérielle du SGBD qui la gère.
- L'indépendance données-traitements permet si elle est atteinte :
  - de modifier l'organisation physique (par exemple ajouter un index pour un accès plus rapide) sans modifier le schéma conceptuel ou les programmes d'applications,
  - de modifier le schéma conceptuel (par exemple ajouter un nouveau type d'entité ou d'association) sans modifier les programmes d'applications non concernés par cet modification.
- On parle aussi d'**indépendance logique** et d'**indépendance physique**.

Qui intervient sur une BD ?

- L'**administrateur** (une personne ou une équipe) :
  - Il définit le schéma conceptuel de la BD et le fait évoluer.
  - Il fixe les paramètres de l'organisation physique de façon à optimiser les performances.
  - Il gère les droits d'accès et les mécanismes de sécurité.
- Les **programmeurs d'application** :
  - Ils définissent les schémas externes et construisent les programmes qui alimentent ou exploitent la BD en vue d'applications particulières.
  - Ils utilisent pour cela le langage de bases de données du SGBD, éventuellement couplé avec un langage de programmation classique.
- Les **utilisateurs finaux** :
  - Ils accèdent à la BD au travers des outils construits par les programmeurs d'applications ou pour les plus avertis au travers du langage de requêtes.

Modèle entité-association

Historique et objectifs

- Le modèle **entité-association** (EA) a été proposé par Peter Chen en 1976 sous le nom Entity-Relationship Model.
- C'est un langage graphique destiné à la construction du modèle conceptuel d'une BD, indépendamment du SGBD qui sera utilisé pour gérer celle-ci.
- Il est à la base de nombreuses méthodes de conception de BD dont la méthode Merise.
- Il est bien adapté à la conception d'une BD relationnelle car la traduction entité-association → relationnel est relativement naturelle.

La BD *Réseau de bibliothèques*

- Les concepts du modèle EA seront illustrés, sur une BD décrivant un réseau de bibliothèques (universitaires, municipales, ...) et dont les données concernent :
  - les bibliothèques dont chacune est identifiée par son nom,
  - les exemplaires de livre dont chacun est identifié par sa cote et est conservé par une bibliothèque du réseau,
  - les livres dont sont tirés ces exemplaires, dont chacun est identifié par son ISBN (numéro international),
  - les personnes dont chacune est identifiée par son nom et son prénom,
  - les auteurs des livres qui sont des personnes,
  - les abonnés à ce réseau de bibliothèques qui sont des personnes,
  - les directeurs des bibliothèques de ce réseau, qui sont des personnes,

  ...

Entités et associations

- Le modèle EA repose sur deux concepts principaux :
  - les **entités** du monde décrit par une BD,
  - les **associations** entre ces entités.
- Une entité est décrite par ses **attributs**. Par exemple, une personne décrite par son nom, son prénom, son âge et la ville dans laquelle elle habite.
- Une entité est identifiée par sa **clé** : les valeurs d'une partie de ses attributs. Par exemple, une personne peut être identifiée par son nom et son prénom.
- Une association exprime un lien entre plusieurs entités. Par exemple, l'emprunt d'un livre par une personne.
- Une association peut aussi avoir des attributs. Par exemple, l'emprunt d'un livre par une personne à une certaine date.

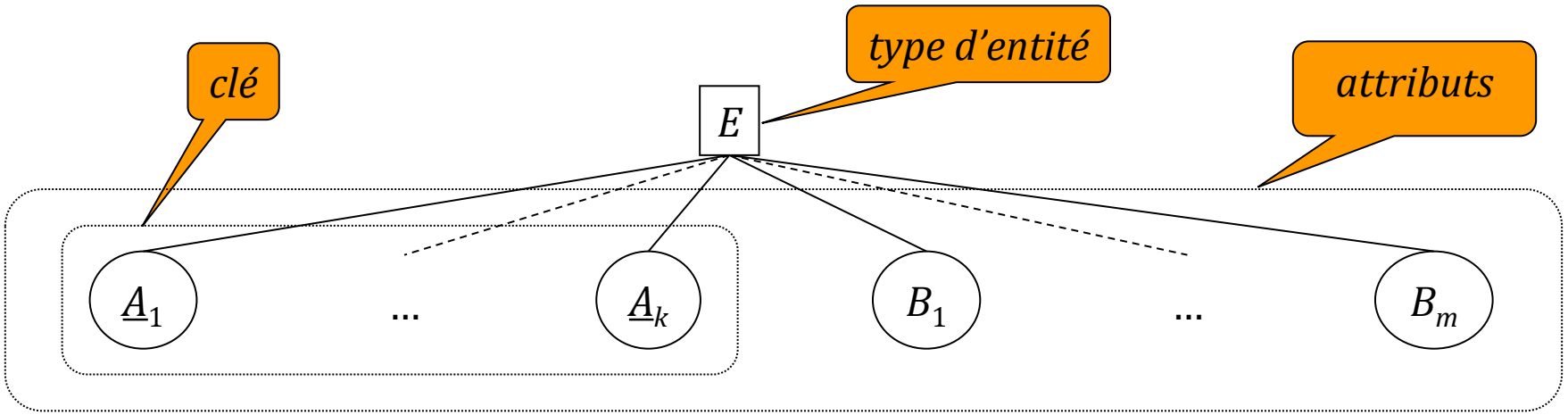
Schéma et extension

- Le **schéma** EA d'une BD est composé de l'ensemble des définitions des types d'entité et d'association auxquelles doivent appartenir les instances de cette BD :
  - toute entité a un type,
  - toute association a un type.
- On appelle **extension** d'un type d'entité ou d'association à un instant donné, l'ensemble des instances de ce type enregistrées dans la BD à cet instant.

Type d'entité (1)

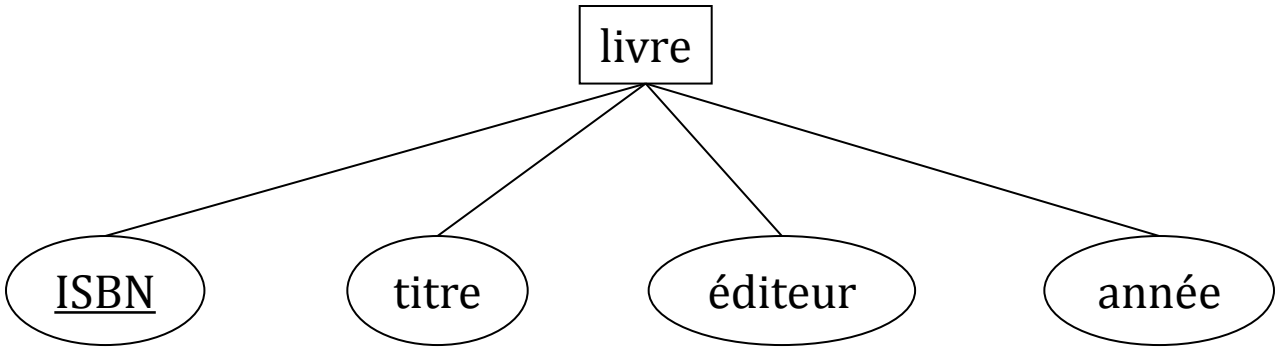
- Un type d'entité est défini par :
  - son **nom**,
  - ses attributs,
  - sa **clé** : les attributs dont les valeurs identifient une entité de ce type.
- Un attribut est défini par :
  - son nom,
  - le type de ses valeurs.
- Pour raison de simplicité :
  - on ne spécifiera pas les types des valeurs des attributs,
  - on considérera que ces valeurs sont atomiques (comme dans le modèle relationnel).

Type d'entité (2)



- Une entité instance de E a pour valeur le n-uplet $(v_1, ..., v_k, w_1, ..., w_m)$ tel que :
  - pour i de 1 à k, v_i est une valeur de l'attribut A_i,
  - pour i de 1 à m, w_i est une valeur de l'attribut B_i,
  - il n'existe pas d'autre entité de E ayant la clé $(v_1, ..., v_k)$.

Type d’entité (3)

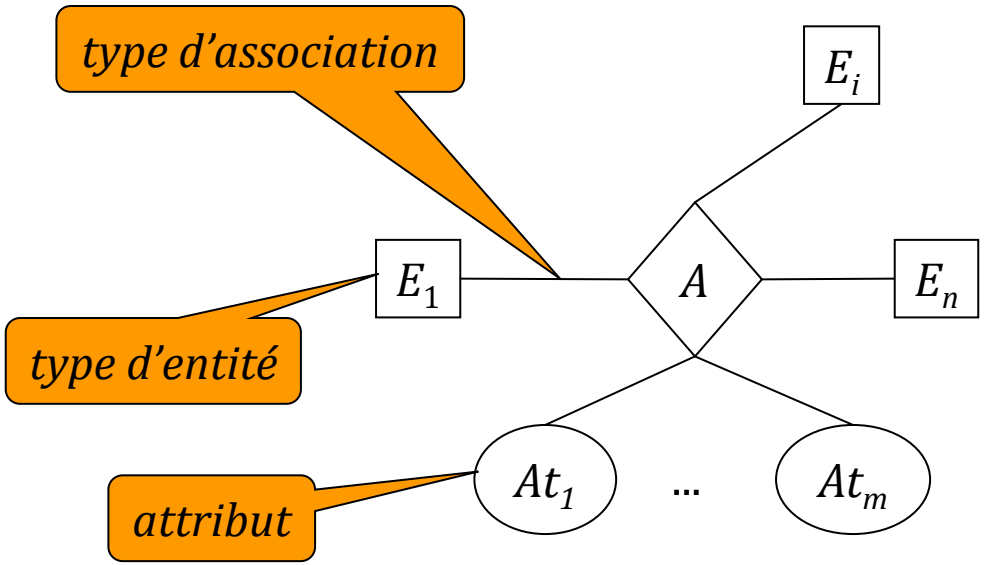


- Un livre est décrit par son ISBN (numéro international), son titre, le nom de son éditeur et son année d'édition.
- Un livre est identifié par son ISBN, ce qui signifie que dans la BD *réseau_bibliothèques*, il n'y a pas deux livres qui ont le même ISBN (et d'ailleurs dans le monde !).

Type d'association (1)

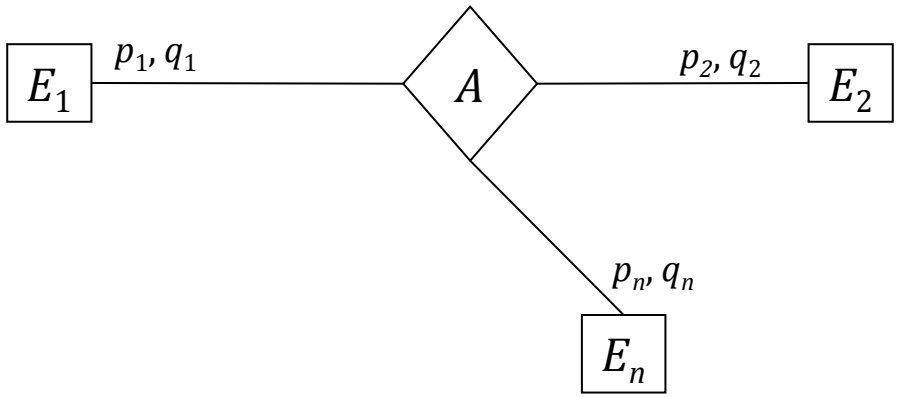
- Un type d'association est défini par :
  - son nom,
  - les noms des types d'entités dont les instances peuvent **participer** aux associations de ce type,
  - sa **cardinalité** qui définit pour chaque type d'entité participant combien de ses instances peuvent participer à une association de ce type,
  - sa **arité** qui définit combien de types d'entités participent à ce type d'association,
  - éventuellement le **rôle** (un nom) que jouent dans une association de ce type, les entités qui y participent,
  - ses attributs.
- On distingue deux types particuliers d'association :
  - les **entités faibles**,
  - les **associations isa**.

Type d'association (2)



- Une association instance de A a pour valeur le n-uplet (e_1, ..., e_n, v_1, ..., v_m) tel que :
  - pour i de 1 à n, e_i est une instance du type d'entité E_i,
  - pour i de 1 à m, v_i est une valeur de l'attribut At_i,
  - il n'existe pas d'autre association de A associant les entités e_1, ..., e_n.

Cardinalité d'un type d'association

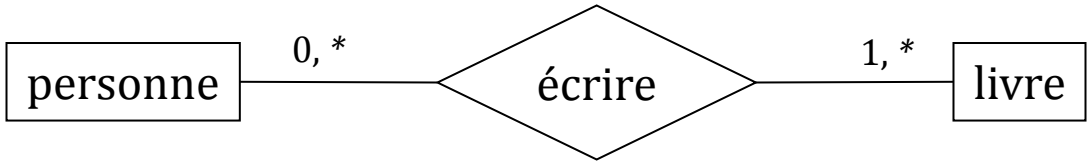


- $p_i \geq 0$ et $q_i \geq p_i$
- Dans toute extension de A, une même entité ne peut pas apparaître moins de p_i fois et plus de q_i fois en position i.

Désignation des entités et des associations

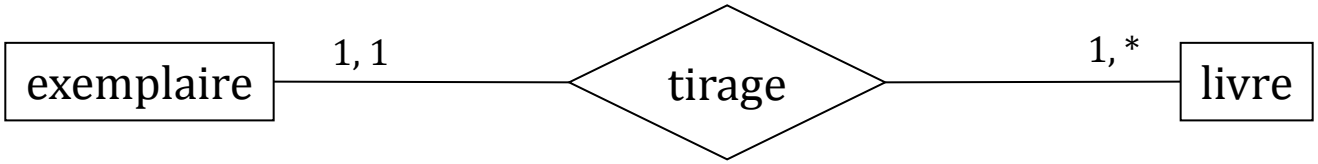
- Il est habituel de désigner un type d'entité par un substantif et une association :
  - soit par un verbe, quand elle exprime une action
  - soit par un substantif quand elle exprime un état
- Par exemple :
  - les type d'entité **livre**, **personne**, **bibliothèque**, **ville**,
  - le type d'association **écrire** qui exprime qu'un livre **a été écrit** par une personne et qu'une personne **a écrit** un livre.
  - le type d'association **localisation** qui exprime qu'un bibliothèque est localisée dans une **ville**.

Type d'association binaire *-*



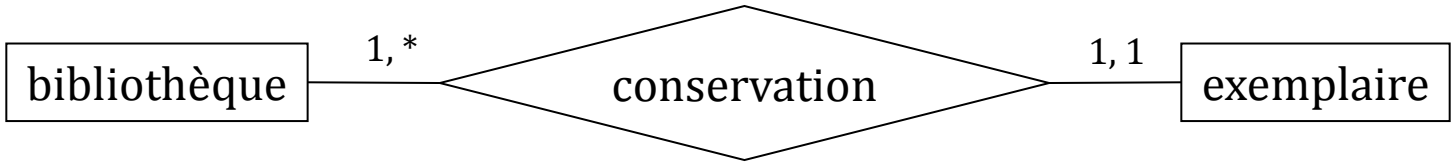
- * représente un entier positif ou nul.
- Une personne peut avoir écrit un ou plusieurs livres : elle peut donc apparaître 0, 1 ou plusieurs fois dans une extension de l'association **écrire**.
- Un livre a été écrit par au moins une personne : il doit donc apparaître 1 ou plusieurs fois dans une extension de l'association **écrire**.

Type d'association binaire 1-*



- Un exemplaire est tiré d'un et d'un seul livre.
- Un livre a été tiré en un ou plusieurs exemplaires.

Type d'association binaire 1-*

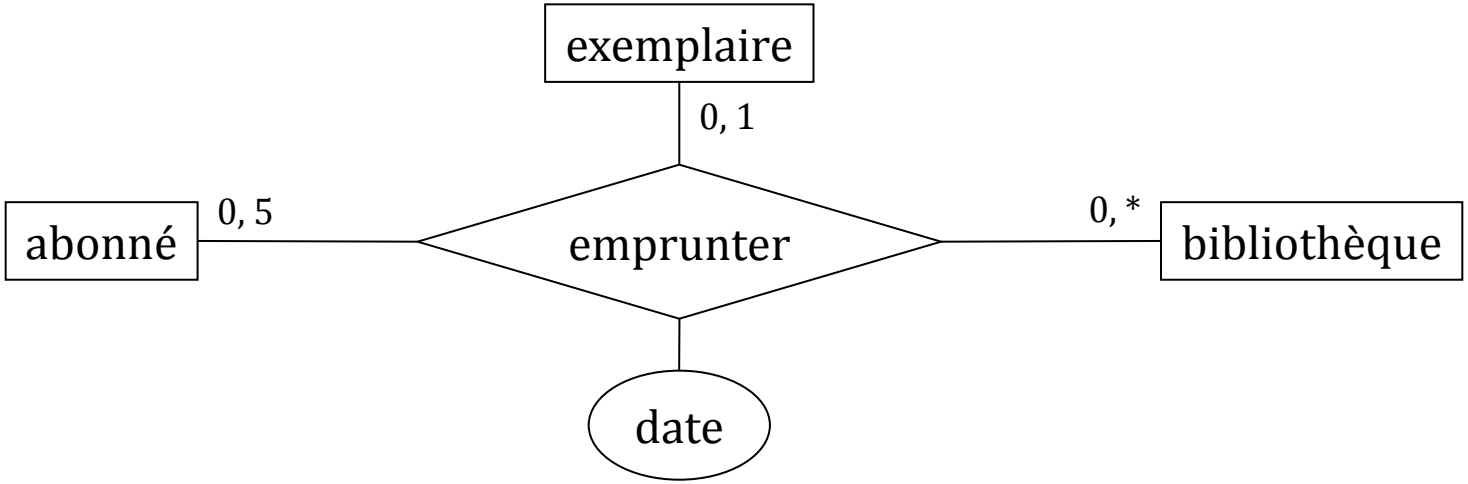


- Une bibliothèque conserve un ou plusieurs exemplaires de livres.
- Un exemplaire d'un livre est conservé dans une et une seule bibliothèque.

Type d'association binaire 1-1

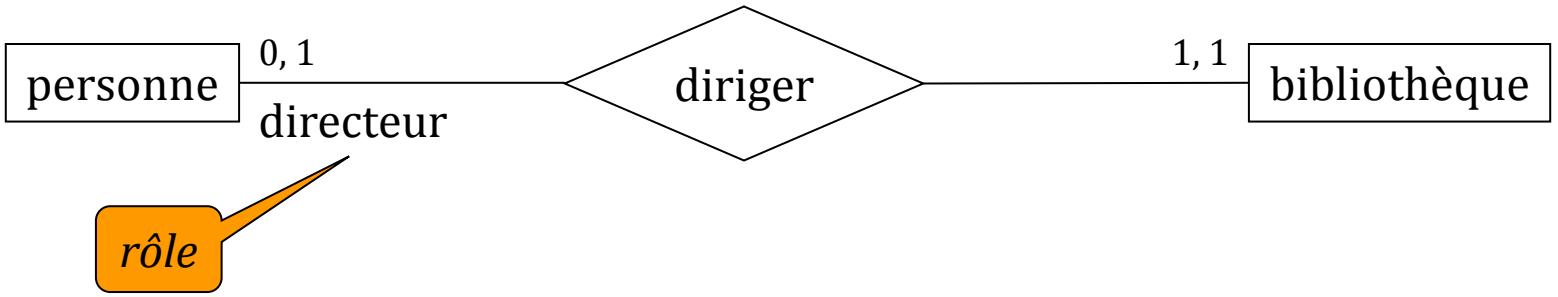
- Une personne peut diriger une bibliothèque au plus.
- Une bibliothèque est dirigée par une et une seule personne.

Type d'association ternaire avec attribut



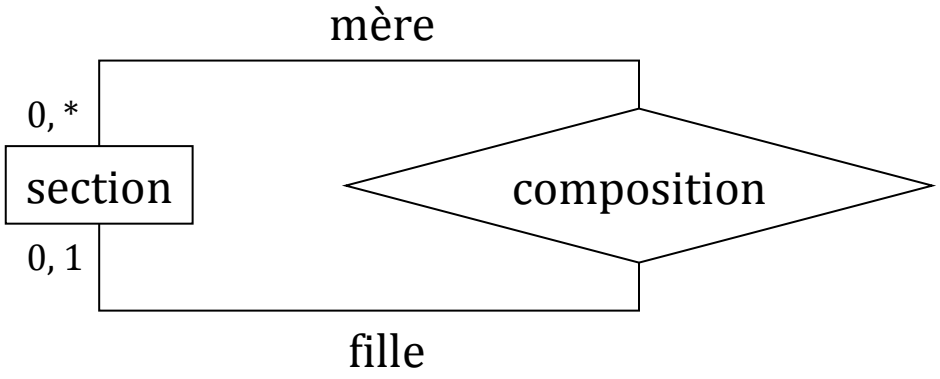
- L'emprunt d'un exemplaire de livre est une action matérialisée par un formulaire créé à la date de l'emprunt et supprimé lorsque l'exemplaire emprunté est rendu.
- Un abonné au réseau peut emprunter un exemplaire d'un livre dans une bibliothèque du réseau (qui peut être différente de celle qui conserve cet exemplaire), mais il ne peut pas avoir plus de 5 emprunts en cours.
- Un exemplaire de livre ne peut apparaître que dans un seul acte d'emprunt.

Type d'association avec rôle (1)



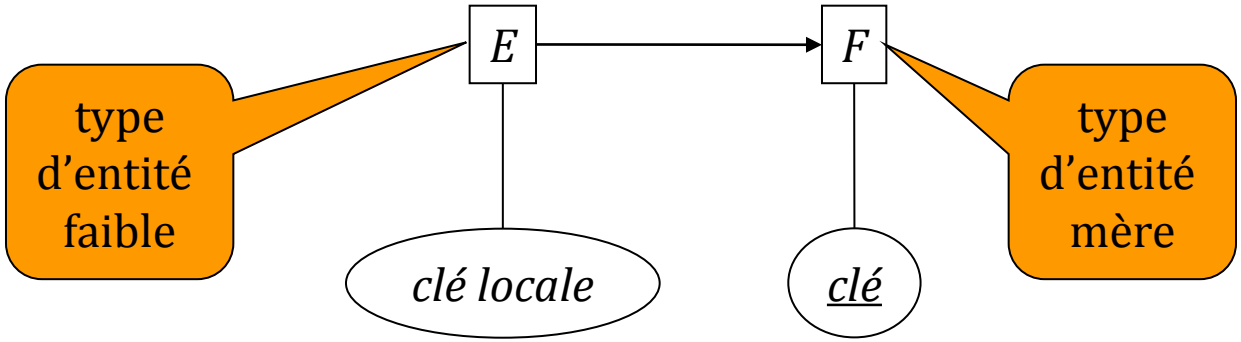
- Dans une association (p, b) de type diriger la personne p joue le rôle de directeur.

Type d'association avec rôle (2)



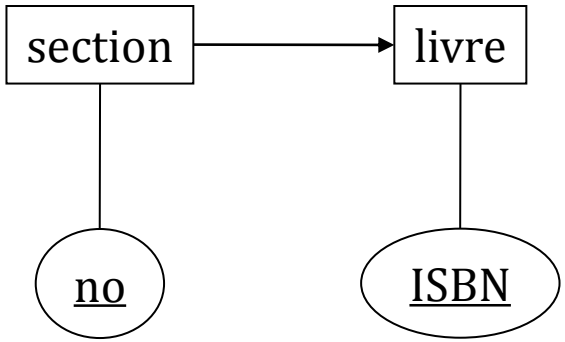
- Lorsqu'un même type d'entité participe plusieurs fois à un type d'association, il est nécessaire d'indiquer le rôle que jouent ses instances dans chaque cas.
- Une section d'un livre peut être composée de sections : une section composée joue le rôle de mère, une section composante joue le rôle de fille.
- Une section ne peut apparaître qu'une seule fois comme section fille.
- Une section peut apparaître plusieurs fois comme section mère.

Type d'entité faible (1)



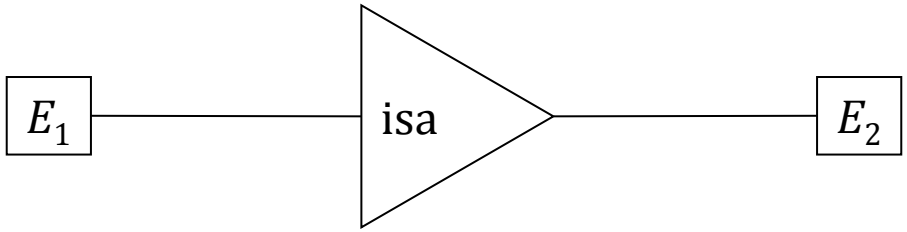
- Une entité faible est une entité qui n'existe que comme composante d'une autre entité : son entité mère. Lorsqu'elle celle-ci est supprimée, elle doit l'être aussi.
- Une entité faible a une **clé locale** qui l'identifie parmi les entités faibles du même type de son entité mère.
- La clé d'une entité faible est donc la concaténation de sa clé locale et de la clé de son entité mère.
- Le processus peut être récursif, mais il doit y avoir une entité mère racine.

Type d'entité faible (2)



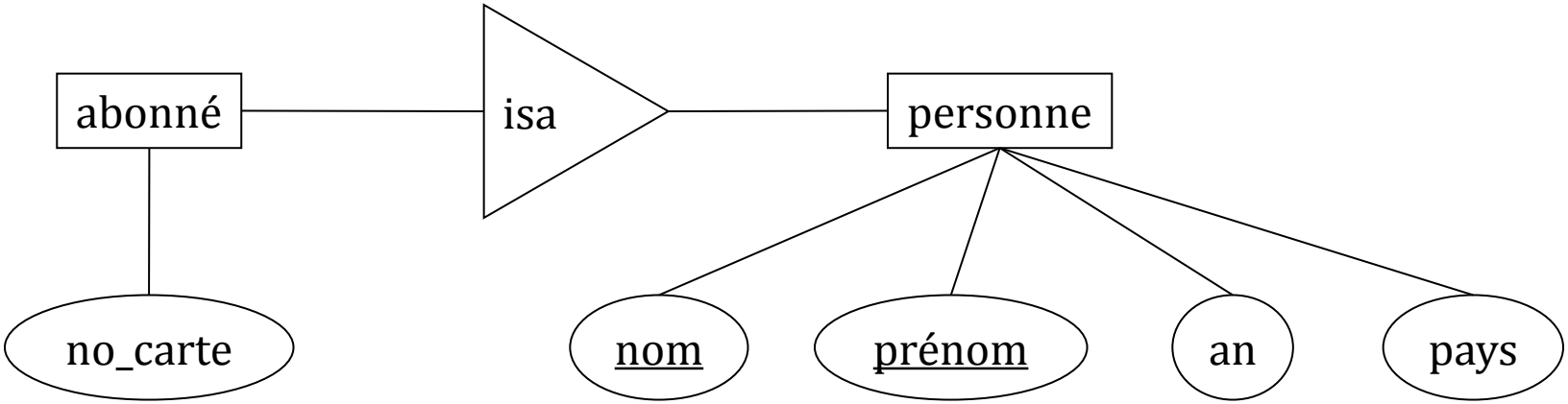
- Une section d'un livre peut être composée d'autres sections.
- Les sections d'un livre sont numérotées. Cette numérotation est locale à ce livre : deux sections de deux livres différents peuvent avoir le même numéro.
- Une entité **section** a donc pour clé locale son numéro (attribut **no**) et pour clé son numéro et l'ISBN (attribut **ISBN**) du livre auquel cette section appartient.

Type d'association *isa* (1)



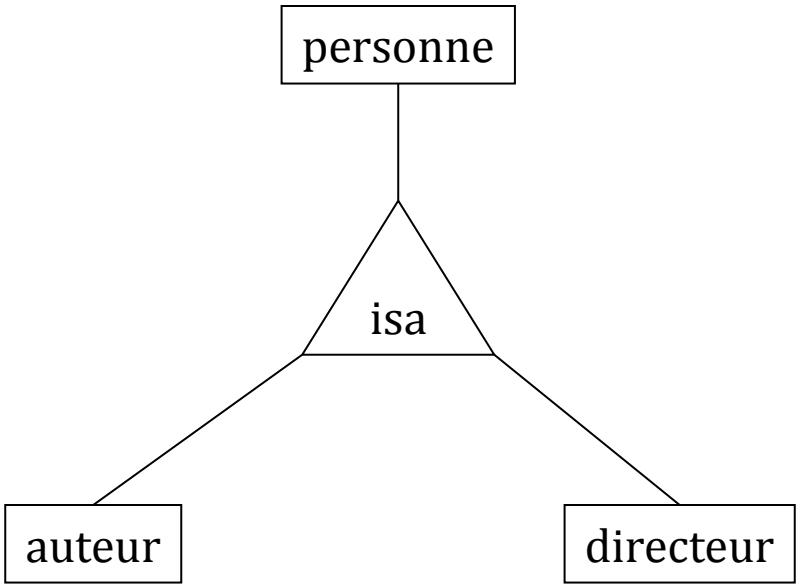
- Une entité de type E_1 possède toutes les propriétés d'une entité de type E_2 plus certaines autres.
- On dit que E_1 est une **spécialisation** de E_2 ou que E_2 est une **généralisation** de E_1.

Type d'association *isa* (2)



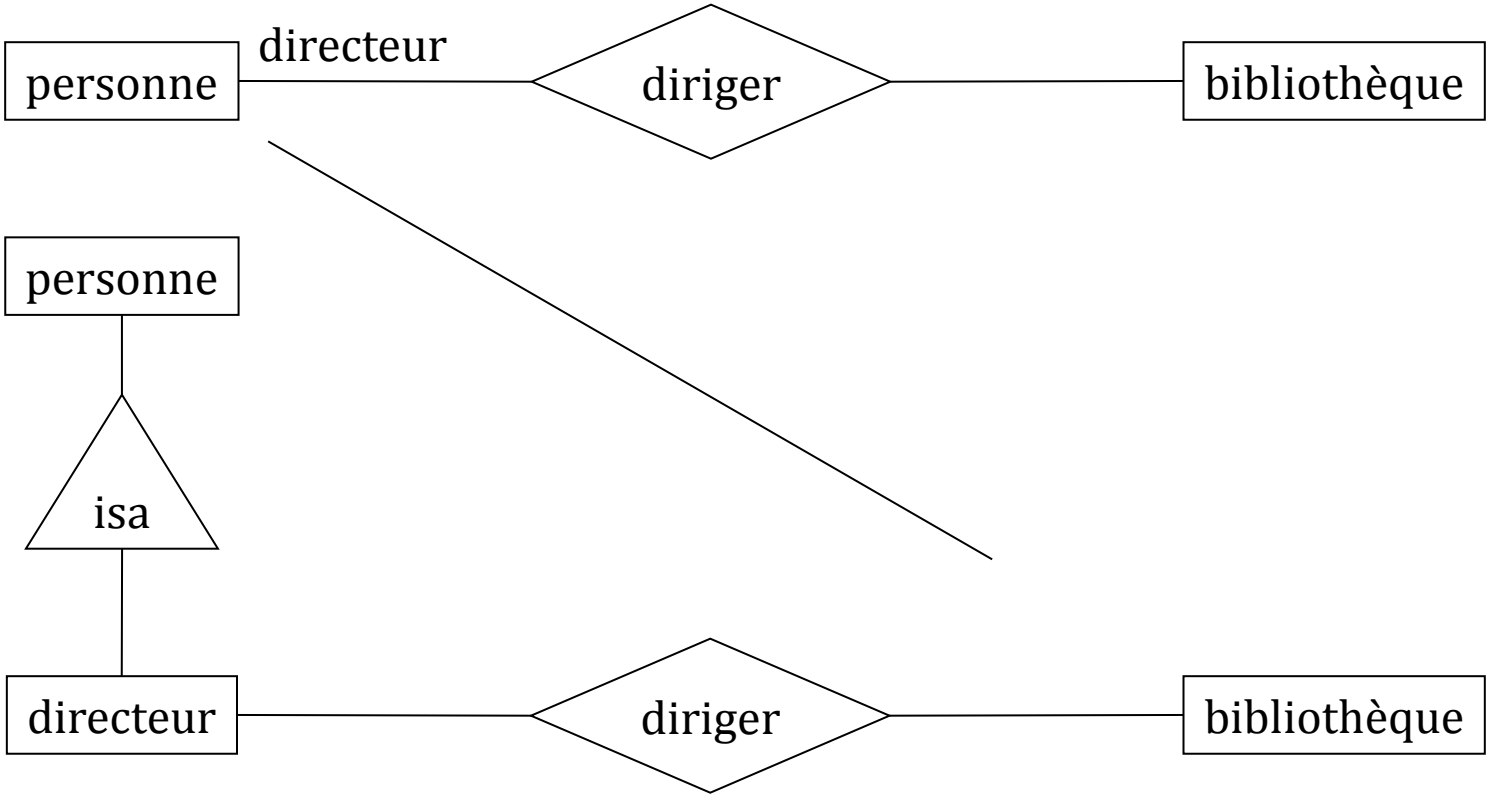
- Une personne est décrite par son nom, son prénom, son année de naissance et son pays d'origine.
- Une personne est identifiée par son nom et son prénom.
- Un abonné (d'une bibliothèque) est une personne :
  - il a donc un nom, un prénom, une année de naissance et un pays d'origine,
  - son nom et son prénom l'identifient.
- Un abonné a de plus un numéro de carte.

Type d'association *isa* (3)



- Un auteur et un directeur de bibliothèque sont des personnes.

Généralisation vs rôle



Modèle relationnel

Introduction

- Proposé par E.F. Codd d'IBM en 1969, le modèle relationnel a fait l'objet d'un grand nombre de travaux de recherche qui, depuis le début des années 1980, ont débouché sur des produits commerciaux ou libres :
  - DB2 d'IBM, Oracle, Sybase, Access ou SQL-Server de Microsoft, PostgreSQL, MySQL...
- Le succès du modèle relationnel est dû à :
  - sa **simplicité** pour l'utilisateur : une BD est vue comme un ensemble de tables,
  - ses **fondements théoriques** : l'algèbre relationnelle et la logique des prédicats.

Constitution d'une BD relationnelle

- Une BD relationnelle est constituée par :
  - un ensemble de **domaines**,
  - un ensemble de **relations**,
  - un ensemble de **contraintes d'intégrité**.

Domaines

- Un domaine est un ensemble de valeurs atomiques.
- On distingue :
  - les domaines **prédéfinis** :
    - l'ensemble des chaînes de caractères (texte),
    - l'ensemble des nombres entiers (entier),
    - l'ensemble des booléens :
      - `booléen` = {vrai, faux}
    - l'ensemble des dates

    ...
  - les domaines **définis** :
    - en **extension**, c.-à-d. en énumérant les valeurs :
      - `couleur` = {"rouge", "vert", "bleu", "jaune"}
    - en **intention**, c.-à-d. en spécifiant la formule que doit vérifier chaque valeur :
      - `mois` = $\{m \mid m \in \text{entier} \wedge 1 \leq m \leq 12\}$

Relations

- Une **relation** R est un sous-ensemble du produit cartésien de n domaines $D_1, \ldots, D_n$:
  - $R \in D_1 \times \ldots \times D_n$
- Une relation est définie par son **nom**, par son **schéma** et par son **extension**.

Schéma d'une relation (1)

- Le **schéma d'une relation** définit les domaines sur lesquels elle est construite et donne un nom à ces domaines.
- Nous noterons :
  - $(A_1: D_1, ..., A_n: D_n)$

  le schéma d'une relation où
  - D_i est un nom de domaine,
  - A_i est un **nom d'attribut** qui indique le rôle du domaine D_i dans la relation.
  - Les noms d'attributs d'un même schéma doivent être tous différents.
- Par exemple :
  - `(nom: texte, âge: entier, marié: booléen)`

  est le schéma d'une relation :
  - construite sur les domaines `texte`, `entier` et `booléen`,
  - qui peut représenter un ensemble de personnes décrites respectivement par leur nom, leur âge et leur état civil (marié ou non marié).

Schéma d'une relation (2)

- Il est souvent pratique de noter à la fois le nom d'une relation et son schéma. La notation :
  - $R(A_1: D_1, \ldots, A_n: D_n)$

  désignera une relation :
  - de nom R,
  - et de schéma $(A_1: D_1, \ldots, A_n: D_n)$.
- Lorsque l'indication des domaines n'est pas requise, un schéma de relation pourra être noté :
  - $(A_1, \ldots, A_n)$

Extension d'une relation

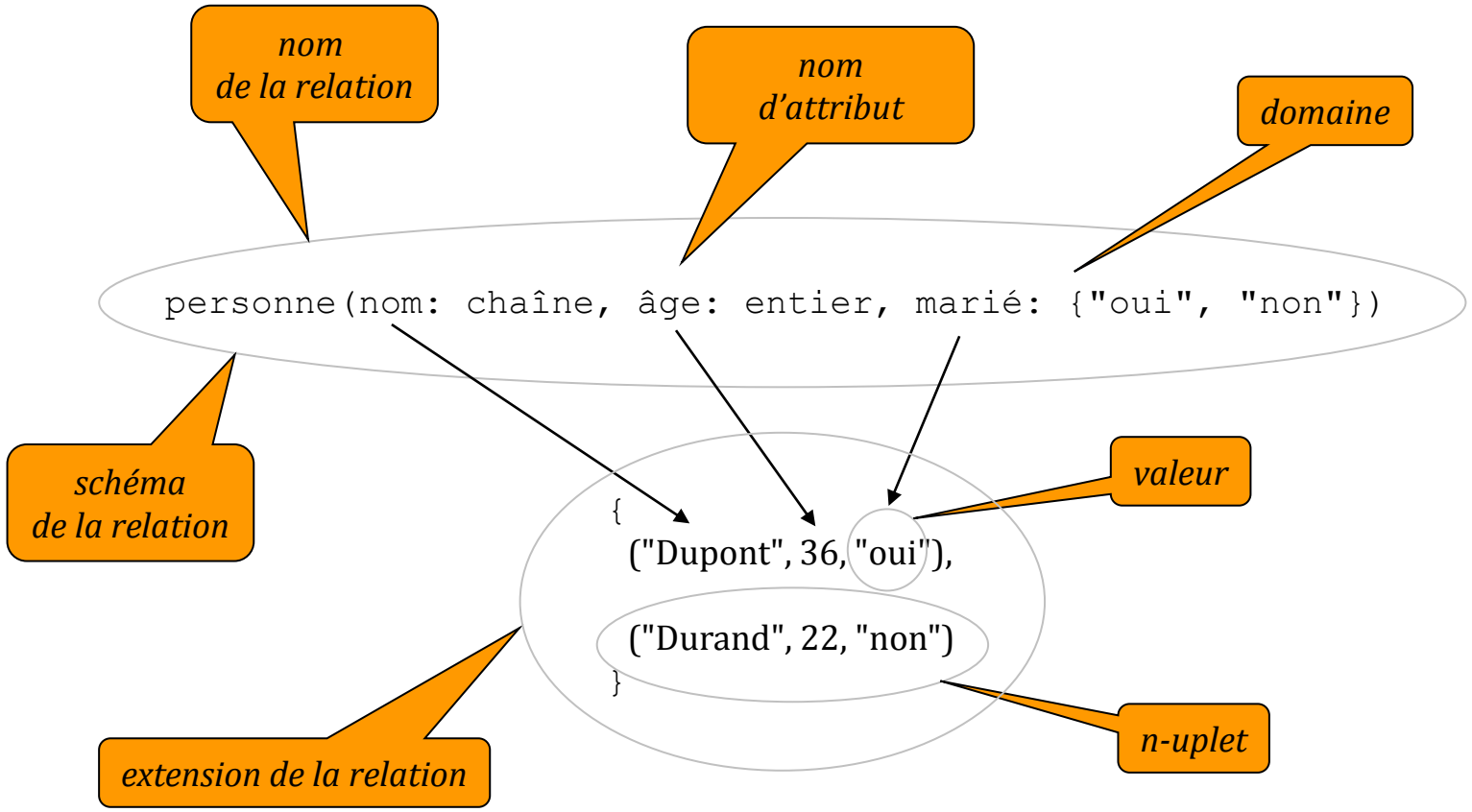
- L'extension d'une relation de schéma :
  - $(A_1: D_1, ..., A_n: D_n)$

  est un ensemble de nuplets notés :
  - soit $(v_1, ... , v_n)$ (dans l'ordre du schéma),
  - soit $\{A_1 = v_1, ... , A_n = v_n\}$ (dans un ordre quelconque).

  et tels que :
  - $v_1 \in D_1, ..., v_n \in D_n$
- L'extension d'une relation est variable au cours de la vie d'une BD.
- Par exemple, une extension possible de la relation de schéma :
  - `(nom: texte, age: entier, marié: booléen)`

  est :
  - {("Dupont", 36, vrai), ("Durand", 22, faux)}

En résumé : anatomie d'une relation



Deux visions d'une relation

- Une relation peut être vue sous deux formes :
  - **tabulaire**,
  - **prédicative.**

Vision tabulaire

- Une relation :
  - $R(A_1: D_1, \ldots, A_n: D_n)$
- peut être vue comme :
  - une table de nom R,
  - possédant n colonnes nommées $A_1, \ldots, A_n$
  - dont chaque ligne représente un n-uplet de l'extension de cette relation.
- Par exemple :

personne		
nom	**âge**	**marié**
Dupont	36	vrai
Durand	22	faux

Vision prédicative

- A toute relation de schéma :
  - $R(A_1: D_1, \ldots, A_n: D_n)$

  il peut être associé un **prédicat** R tel que l'assertion $R\ t$ est vraie si le n-uplet t appartient à l'extension de R et fausse sinon.
- Par exemple, pour l'extension considérée, l'assertion :
  - `personne("Dupont", 36, vrai)`

  est vraie.

Valeurs nulles

- Il peut arriver que certaines informations soient inconnues ou non pertinentes.
- Par exemple, une conférence dont la date n'est pas encore fixée ou bien le nombre de couleurs pour un écran noir et blanc.
- Pour représenter une telle absence d'information on utilise une valeur particulière :
  - la **valeur nulle** que nous noterons _.
- Par exemple, le triplet :
  - ("L'avenir des bases de données", "Paul Durand", _)
- peut indiquer que la date de la conférence « L'avenir des bases de données » donnée par Paul Durand, n'est pas encore connue.

Constituants d'une relation

- Nous appellerons **constituant** d'une relation un sous-ensemble, éventuellement vide, des attributs de cette relation.
- Par exemple, l'ensemble d'attributs `{nom, âge}` est un constituant de la relation `personne`.

Entités, associations et clés

- Dans le modèle relationnel :
  - les entités sont identifiées au travers des **clés primaires**,
  - les associations sont décrites au moyen des **clés étrangères.**

Clés candidates et clé primaire (1)

- Un constituant *X* est la **clé candidate** d'une relation *R* si :
  - pour chaque *n*-uplet de *R*, la valeur de *X* identifie de façon unique ce *n*-uplet,
  - aucun attribut de *X* ne peut être supprimé sans détruire la propriété précédente.
- Une relation peut avoir une ou plusieurs clés candidates : l'une est choisie comme **clé primaire**.

Clés candidate et clé primaire (2)

- Soit par exemple, la relation :
  - `personne(nom, prénom, numss, pays)`

  où `numss` est le numéro de sécurité sociale.
- Si une personne est identifiée :
  - soit par son nom et son prénom (et donc qu'elle ne l'est pas par son nom seul ou son prénom seul),
  - soit par son numéro de sécurité sociale,

  les clés candidates de la relation `personne` sont :
  - `{nom, prénom}`
  - `numss`
- `numss` pourra être choisi comme clé primaire.
- Par convention, dans le schéma d'une relation, on souligne les attributs de la clé primaire :
  - `personne(nom, prénom, numss, pays)`

Clés étrangères (1)

- Un constituant Y d'une relation R_1 est une clé étrangère de R_1 s'il existe une relation R_2 possédant une clé primaire X et que Y a pour domaine l'ensemble des valeurs de X.
- On dit que Y **réfère** la relation R_2.

Clés étrangères (2)

- Soit par exemple les relations :
  - `personne(nom, prénom, âge)`
  - `livre(cote, titre, nom_auteur, prénom_auteur)`
- Pour indiquer que l'auteur d'un livre est une personne de la BD, on déclare que le constituant :
  - `{nom_auteur, prénom_auteur}`

  est une clé étrangère de la relation `livre` qui réfère la relation `personne`.

Contraintes d'intégrité (1)

- Les contraintes d'intégrité d'une BD relationnelle peuvent s'exprimer par :
  - l'appartenance des valeurs d'attributs à des domaines,
  - la définition des clés,
  - la normalisation des relations,
  - un ensemble d'assertions,
  - des conditions associées aux opérations de mise à jour.

Contraintes d'intégrité (2)

- Concernant les clés, deux formes d'intégrité jouent un rôle important :
  - L'**intégrité d'entité** qui est vérifiée si les valeurs des attributs de la clé primaire ne sont pas nulles.
  - L'**intégrité référentielle** qui est vérifiée si chaque valeur d'une clé étrangère *Y* :
    - soit existe comme valeur de la clé primaire d'un *n*-uplet de la relation que *Y* réfère,
    - soit est nulle.

Exemple : la BD plus8000

- Cette BD concerne les premières ascensions des 14 sommets de plus de 8000 m.
- Nous l'appellerons : *plus8000*.

Quelques photos !

Everest, 8848 m
(auteur : Pavel Novak)

(photos extraites de Wikimedia Commons)

Nanga Parbat, 8126 m
(auteur : Adam Jacob Muller)

Broad Peak, 8047 m
(auteur : Kogo)

Les faits (1)

- On veut représenter les faits suivants :
  - Le **nom**, l'**altitude**, l'**année** de la 1ère ascension de chaque sommet et l'orientation de la **face** (ou de l'arête) empruntée. Un sommet est identifié par son nom.
  - La **localisation** d'un sommet, c'est à dire le nom du ou des **pays** dans lequel il se trouve. Un sommet peut se trouver dans plusieurs pays quand il appartient à la frontière de chacun de ces pays.
  - Le **nom**, le **prénom** et le **pays** de chaque **grimpeur** ayant réalisé la 1ère ascension d'un sommet de plus 8000 m. Un grimpeur est identifié par son nom et son prénom.
  - Le **nom** et le **prénom** du grimpeur et le nom du sommet gravi pour chaque 1ère **ascension**.

Les faits (2)

- Par exemple :
  - L'Everest a une altitude de 8848m, sa 1ère ascension a été réalisée par son arête SE, en 1953.
  - L'Everest est situé sur la frontière du Népal et de la Chine (Tibet).
  - Edmund Hillary est un grimpeur de Nouvelle-Zélande.
  - Tenzing Norgay est un grimpeur du Népal.
  - Edmund Hillary a réalisé la 1ère ascension de l'Everest.
  - Tenzing Norgay a réalisé la 1ère ascension de l'Everest.

Domaines et relations

- Domaine
  - `orientation = {"N", "S", "O", "E", "NO", "SO", "NE", "SE"}`
  - `pays_himalayen = {"Chine", "Inde", "Népal", "Pakistan"}`
  - `pays = {"Afghanistan", "Afrique du Sud", "Albanie", …}`
- Relations
  - `sommet(nom_sommet: texte, altitude: entier, année: entier, face: orientation)`
  - `localisation(nom_sommet: texte, pays: pays_himalayen)`
  - `grimpeur(nom: texte, prénom: texte, pays: pays)`
  - `ascension(nom_grimpeur: texte, prénom_grimpeur: texte, nom_sommet: orientation)`

Clés (1)

- Relation `sommet(nom, altitude, année, face)` :
  - un sommet est identifié par son nom : l'attribut `nom` est donc la clé primaire.
- Relation `localisation(nom_sommet, pays)` :
  - un sommet peut être sur une ligne frontière et donc appartenir à plusieurs pays : le constituant {`nom_sommet`, `pays`} est donc la clé primaire
  - un sommet localisé doit être décrit dans la relation `sommet` : l'attribut `nom_sommet` est donc une clé étrangère qui réfère l'attribut `nom` de la relation sommet.
- Relation `grimpeur(nom_grimpeur, prénom_grimpeur, pays)` :
  - un grimpeur est identifié par son nom et son prénom : le constituant {`nom`, `prénom`} est donc la clé primaire.

Clés

- Relation `gravir(nom_grimpeur, prénom_grimpeur, nom_sommet)` :
  - plusieurs grimpeurs peuvent avoir réalisé la 1ère ascension d'un sommet : le constituant {`nom_grimpeur`, `prénom_grimpeur`, `nom_sommet`} est donc la clé primaire.
  - un grimpeur ayant réalisé une ascension doit être décrit dans la relation `grimpeur` : le constituant {`nom_grimpeur`, `prénom_grimpeur`} est donc une clé étrangère qui réfère le constituant {`nom`, `prénom`} de la relation `grimpeur`.
  - un sommet sur lequel a été réalisé une ascension doit être décrit dans la relation `sommet` : l'attribut `nom_sommet` est donc une clé étrangère qui réfère l'attribut `nom` de la relation `sommet`.

Vision tabulaire

localisation	
nom_sommet	**pays**
Everest	Népal
Everest	Chine
Nanga Parbat	Pakistan
Broad Peak	Pakistan
Broad Peak	Chine

sommet			
nom	**altitude**	année	face
Everest	8848	1953	SE
Nanga Parbat	8126	1953	N
Broad Peak	8047	1957	O

grimpeur		
nom	**prénom**	**pays**
Hillary	Edmund	Nouvelle-Zélande
Norgay	Tensing	Népal
Bull	Hermann	Autriche
Schmuck	Marcus	Autriche
Wintersteller	Fritz	Autriche
Diemberger	Kurt	Autriche

ascension		
nom_ grimpeur	**prénom_ grimpeur**	**nom_sommet**
Hillary	Edmund	Everest
Norgay	Tensing	Everest
Bull	Hermann	Nanga Parbat
Bull	Hermann	Broad Peak
Diemberger	Everest	Broad Peak
Schmuck	Marcus	Broad Peak
Wintersteller	Fritz	Broad Peak

Vision prédicative

- L'extension de la relation `sommet` est formée de l'ensemble des quadruplets (x, y, z, t) exprimant le fait qu'il y a un sommet de nom x, d'altitude y dont la 1ère ascension a été réalisée l'année z par la face t.
- L'extension de la relation `localisation` est formée de l'ensemble des doublets (x, y) exprimant le fait que le sommet de nom x se trouve dans le pays de nom y.
- L'extension de la relation `grimpeur` est formée de l'ensemble des triplets (x, y, z) exprimant le fait qu'il y a un grimpeur de nom x, de prénom y et de pays z.
- L'extension de la relation `ascension` est formée de l'ensemble des triplets (x, y, z) exprimant le fait que le grimpeur de nom x et de prénom y a réalisé la 1ère ascension du sommet z.

Contraintes d'intégrité

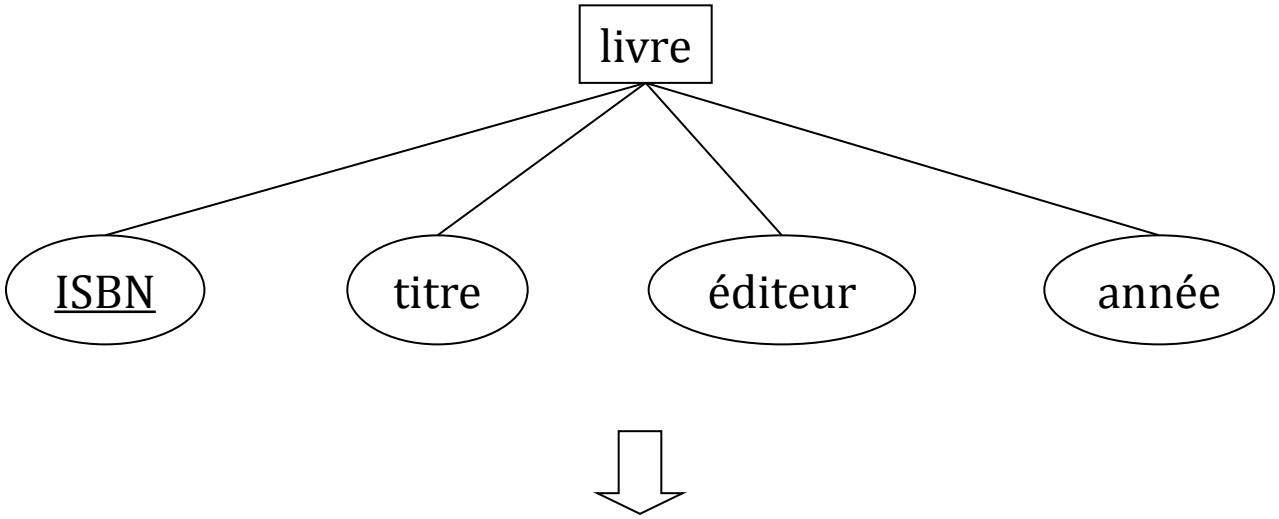
- Ce sont les contraintes autres que celles exprimées par l'appartenance à un domaine ou par une clé :
  - L'altitude d'un sommet doit être supérieure ou égale à 8000 mètres.
  - Tout sommet doit avoir été gravi par au moins un grimpeur.
  - Tout grimpeur doit avoir gravi au moins un sommet.
  - Tout sommet doit avoir une localisation.

Du modèle entité-association au modèle relationnel

Traduction d'un type d'entité (1)

- Un type d'entité est traduit en une relation de même nom, de mêmes attributs et de même clé.

Traduction d'un type d'entité (2)



livre(<u>ISBN</u>, titre, éditeur, année)

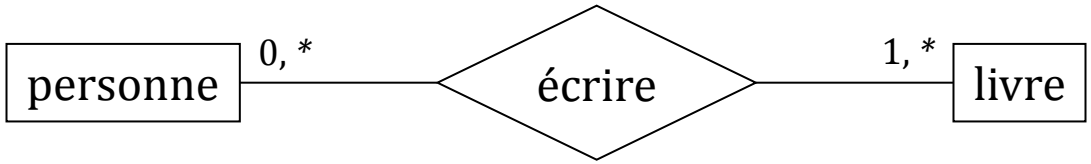
Traduction d'un type d'association (1)

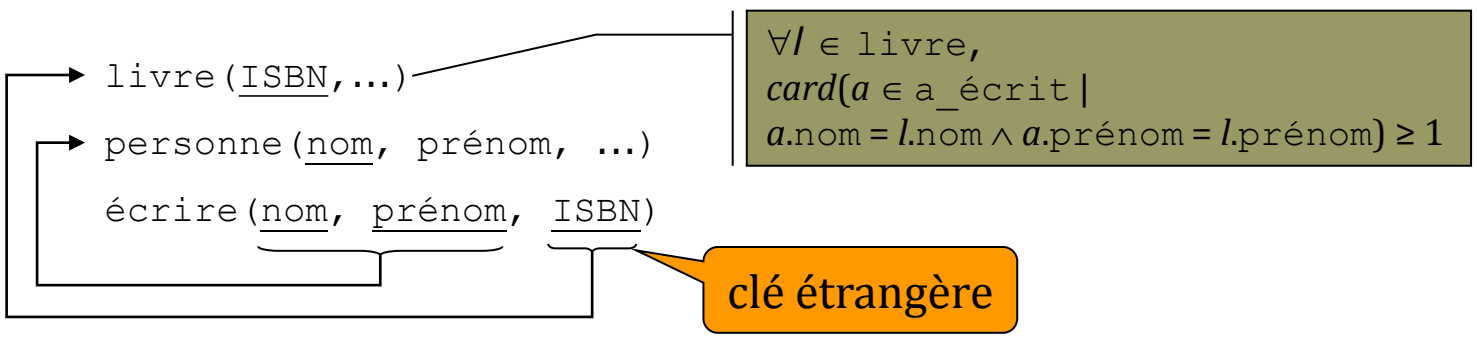
- Un type d'association :
  - $A(E_1, \ldots E_n ; attributs)$

  est traduit en une relation de schéma :
  - $A(\underline{clé(E_1)}, \ldots, \underline{clé(E_n)}, attributs)$

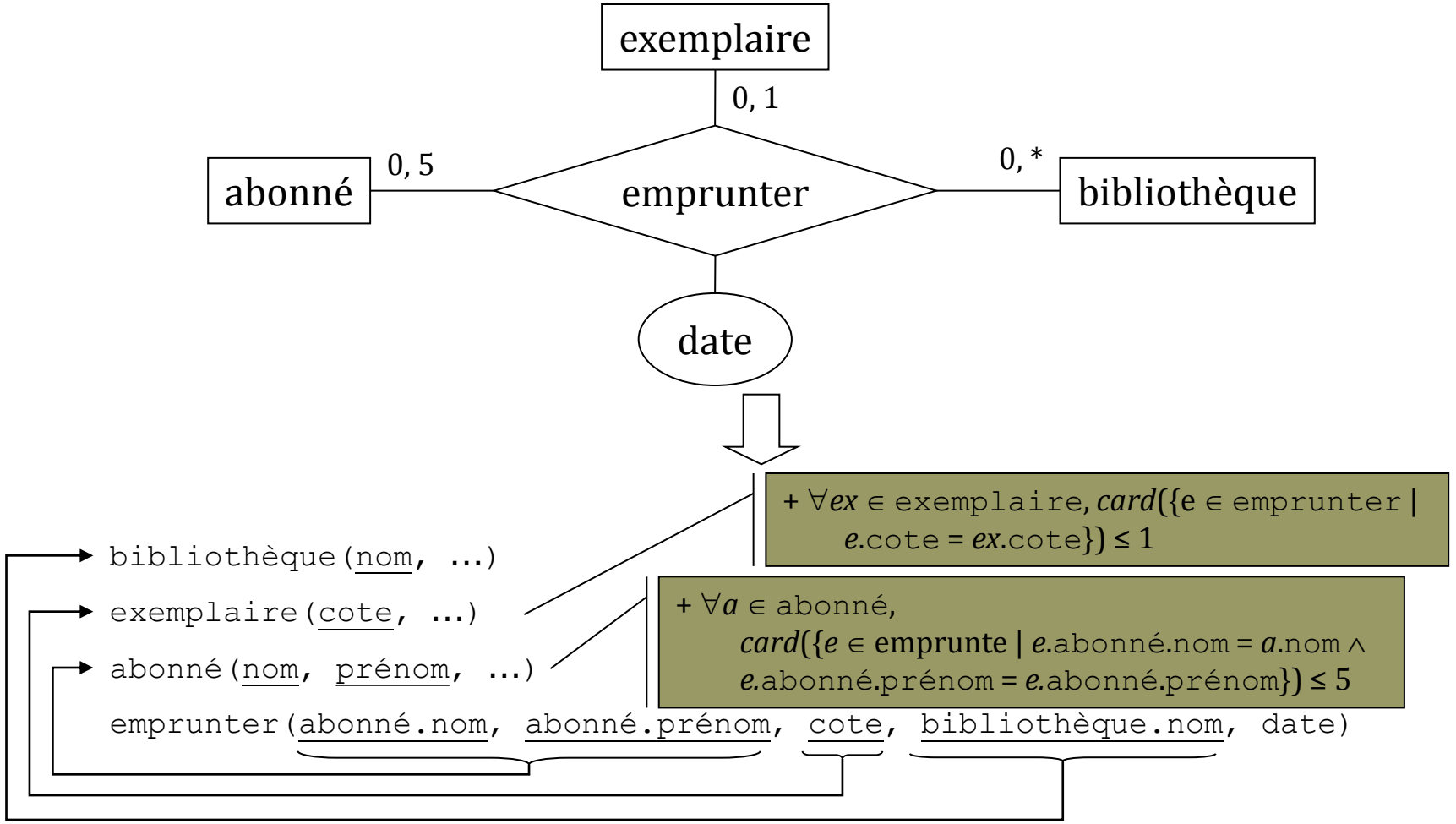
  où $clé(E_1), \ldots, clé(E_1)$ sont des clés étrangères.
- Dans le cas particulier où les cardinalités maximum associées aux types d'entité $E_1, \ldots, E_{n-1}$ sont égales à 1, la traduction peut consister à leur ajouter la clé de la relation E_n, qui est alors une clé étrangère référant la relation E_n.
- Les cardinalités minimales différentes de 0 et les cardinalités maximales différentes de * devront être exprimées sous forme de contraintes d'intégrité de type assertion.
- Afin d'éviter les collisions de noms dues à l'importation des clés, on pourra préfixer les noms des attributs constituant ces clés :
  - soit par le nom de l'entité d'où ils proviennent,
  - soit par le nom du rôle que joue cette entité, s'il est spécifié.

Traduction d'un type d'association (2)

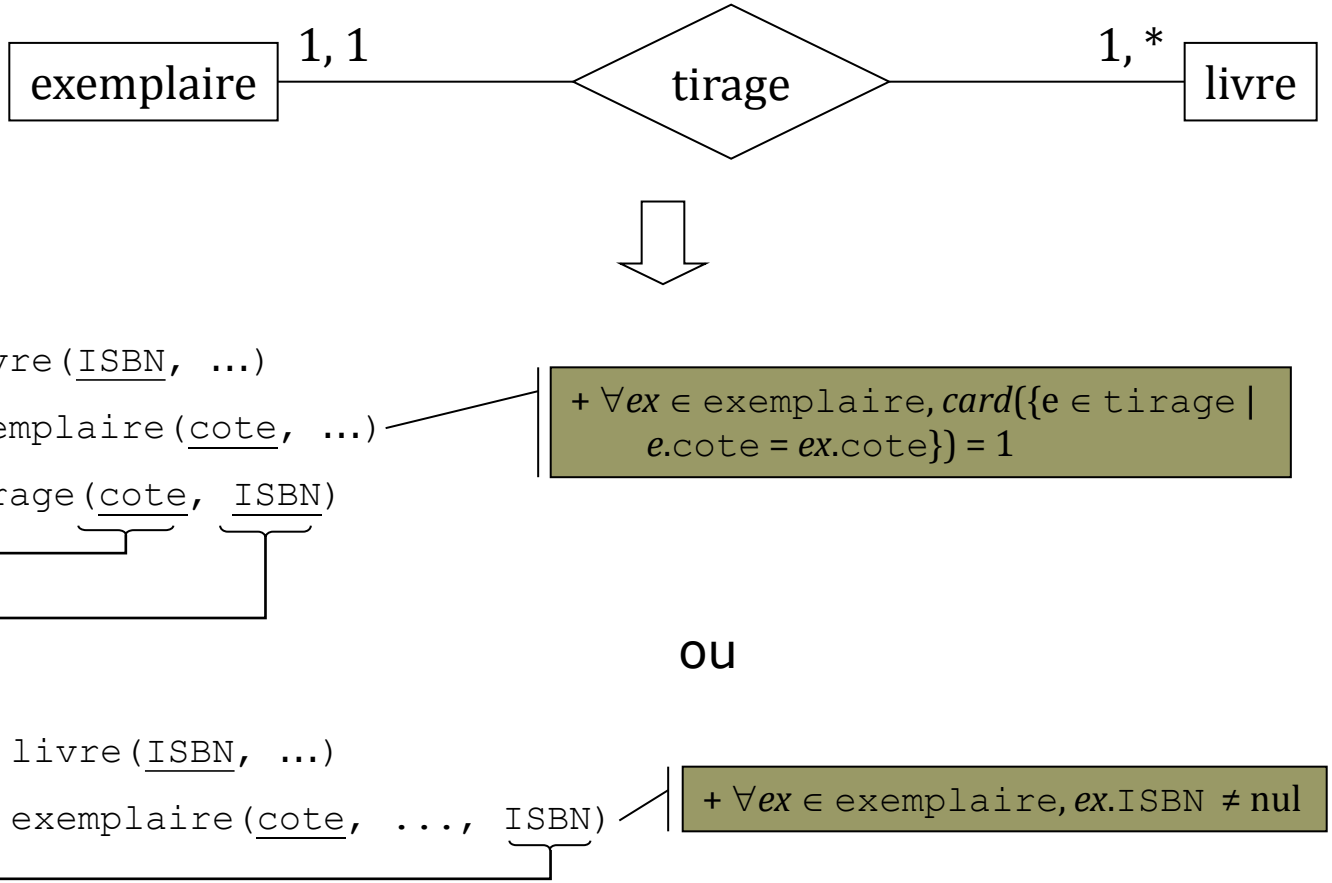




Traduction d'un type d'association (3)



Traduction d'un type d'association (4)



Traduction d'un type d'association (6)

+ $\forall b \in$ bibliothèque,
$card(\{d \in$ diriger $\mid d.$bibliothèque.nom $= b.$nom$\}) = 1$

bibliothèque(<u>nom</u>, ...)

+ $\forall p \in$ personne,
$card(\{d \in$ diriger $\mid d.$directeur.nom $= p.$nom $\wedge$
$d.$directeur.prénom $= p.$prénom$\}) \leq 1$

personne(<u>nom</u>, <u>prénom</u>, ...)

diriger(directeur.nom, directeur.prénom, bibliothèque.nom)

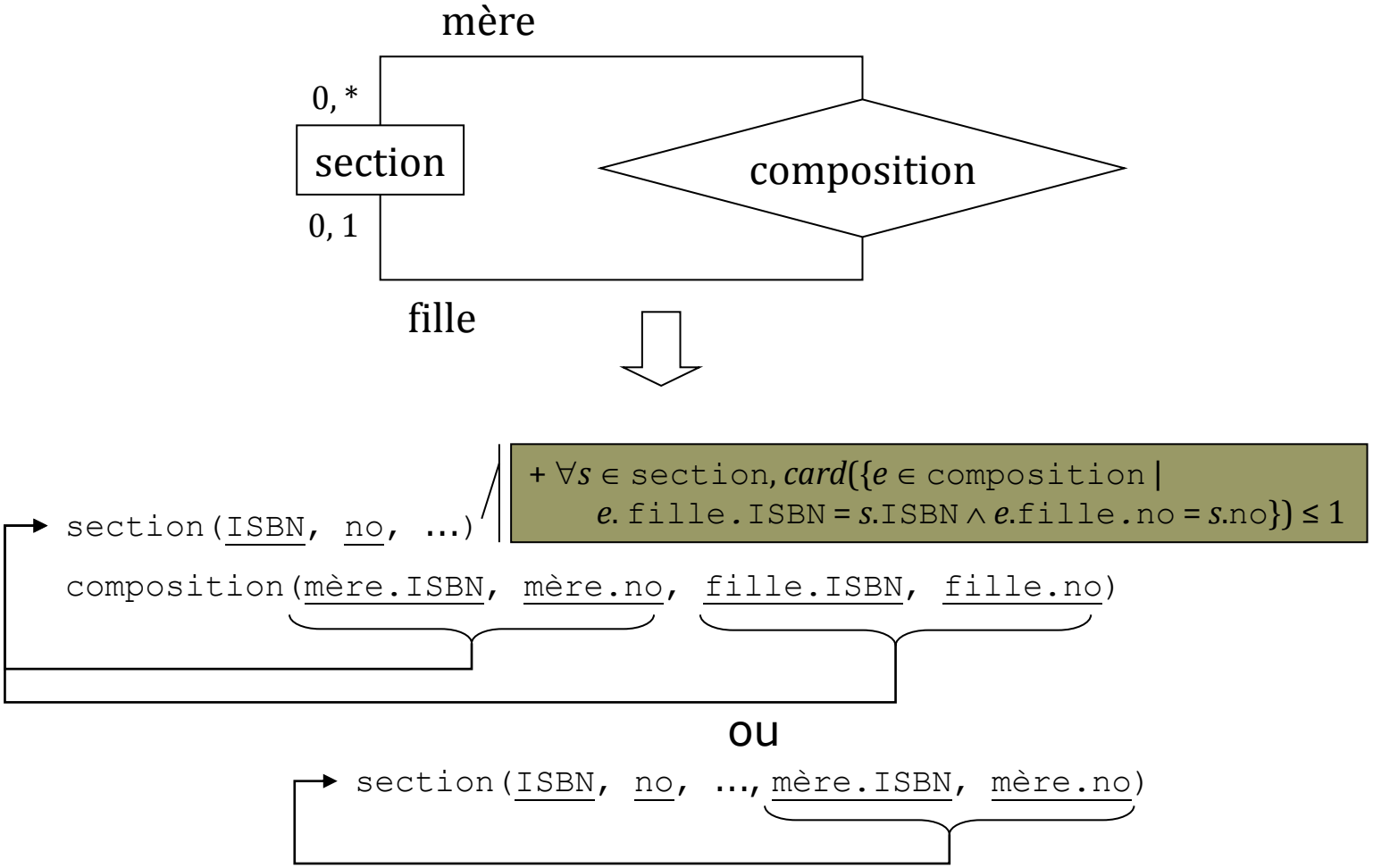
ou

personne(<u>nom</u>, <u>prénom</u>, ...)

bibliothèque(<u>nom</u>, ...,
directeur.nom, directeur.prénom)

+ $\forall b \in$ bibliothèque,
$b.$directeur.nom $\neq$ nul $\wedge$
$b.$directeur.prénom $\neq$ nul

Traduction d'un type d'association (7)



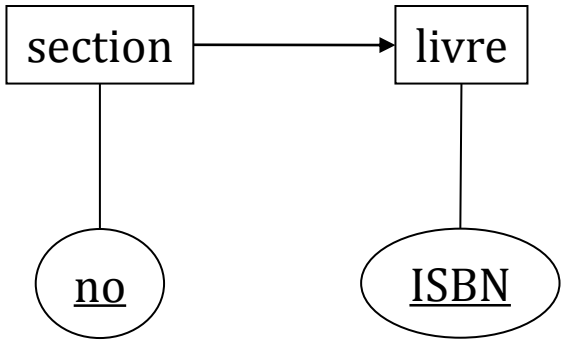
Traduction d'un type d'entité faible (1)

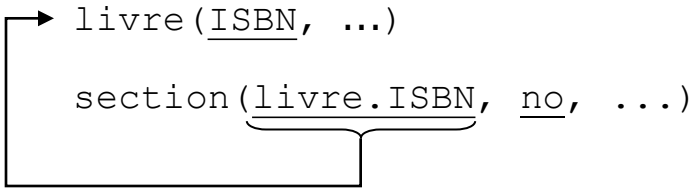
- Un type d'entité faible :
  - $E_1 \rightarrow E_2$

  est traduit en une relation de schéma :
  - $E_1(\underline{clé(E_2)}, \underline{clé\ locale(E_2)}, attributs(E_1))$

  où $clé(E_2)$ est une clé étrangère référant la relation E_1.

Traduction d'un type d'entité faible (2)





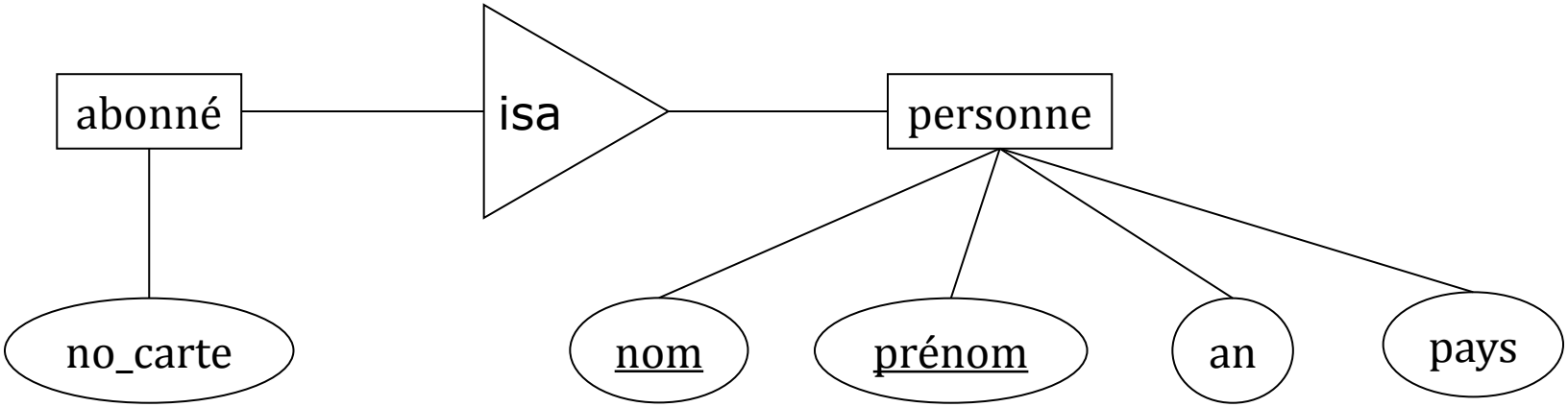
Traduction d'un type d'association *isa* (1)

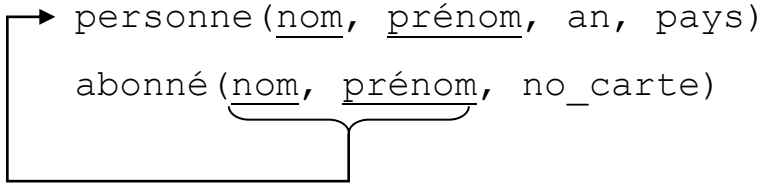
- Un type d'association :
  - E_1 isa E_2

  est traduit en une relation :
  - $E_1(\underline{clé(E_2)}, attributs(E_1))$

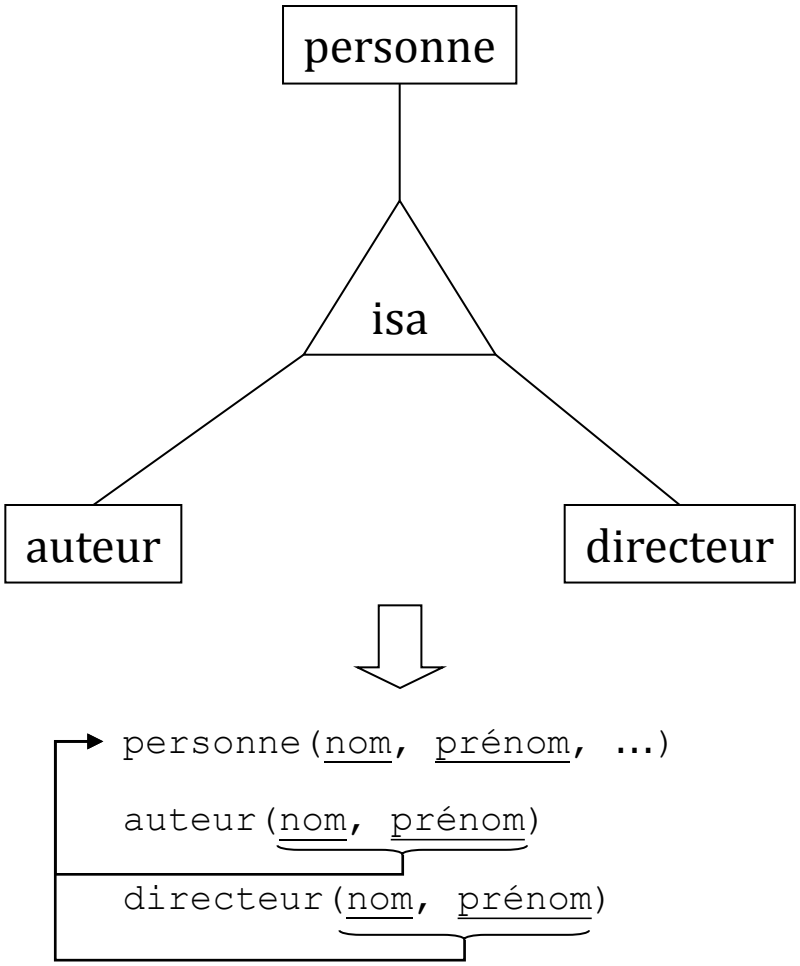
  où $clé(E_2)$ est une clé étrangère référant la relation E_2.

Traduction d'un type d'association *isa* (2)





Traduction d'un type d'association *isa* (3)



Algèbre relationnelle

Objectifs

- L'algèbre relationnelle fournit les opérateurs de base pour manipuler les extensions des relations (ensembles de n-uplets ou tables) d'une BD relationnelle.
- On peut faire un parallèle avec l'arithmétique qui fournit les opérations de base pour manipuler les nombres (addition, soustraction, multiplication et division).
- Tout SGBD relationnel se doit de fournir des implantations efficaces de ces opérateurs implantation de ces opérateurs.
- Une requête SQL est traduite, comme on le verra, en un arbre d'opérateurs de l'algèbre relationnelle.

Expressions (1)

- L'algèbre relationnelle permet de construire des expressions à partir :
  - de noms de relations :
    - relations stockées dans la BD,
    - relations temporaires ;
  - de comparateurs, de connecteurs logiques , d'opérateurs arithmétiques... ;
  - d'opérateurs ensemblistes qui :
    - sélectionnent certaines parties d'une relation,
    - combinent les n-uplets de deux relations.
- Une expression de l'algèbre relationnelle a pour valeur une relation :
  - elle exprime une requête à la BD dont la réponse est la valeur de cette expression.

Expressions (2)

- On notera :
  - p, q, r des expressions de l'algèbre relationnelle,
  - A, B, C des noms d'attributs,
  - *schéma*(r) le schéma de la relation valeur de l'expression r,
  - *ext*(r) l'extension de la relation valeur de l'expression r,
  - $t.A$ la valeur de l'attribut A dans le n-uplet t.
  - *conc* la concaténation de deux n-uplets :
    - $(v_1, \ldots, v_m)$ *conc* $(w_1, \ldots, w_n) = (v_1, \ldots, v_m, w_1, \ldots, w_n)$.

Expressions (3)

- Si R est le nom d'une relation, alors R est une expression telle que :
  - $schéma(R)$ = schéma de la relation désignée par R,
  - $extension(R)$ = extension de la relation désignée par R.

Relations temporaires

- Il est possible de créer des relations temporaires.
- L'instruction :
  - $R := e;$

  crée une relation temporaire de nom R dont le schéma et l'extension sont celles de la relation valeur de l'expression e.

Principaux opérateurs

- Les principaux opérateurs de l'algèbre relationnelle sont les suivants :
  - renommage,
  - union, intersection et différence,
  - produit cartésien,
  - sélection
  - projection,
  - jointure :
    - interne,
    - externe,
    - naturelle
  - division.

Renommage (1)

- L'opérateur de **renommage** (ρ) permet de renommer les attributs d'une relation.
- Soit r une expression telle que :
  - $schéma(r) = (A_1, ..., A_n)$

  l'opérateur ρ est défini par :
  - $schéma(\rho_{B_1, ..., B_n}(r)) = (B_1, ..., B_n)$
  - $ext(\rho_{B_1, ..., B_n}(r)) = ext(r)$

Renommage (2)

$\rho_{nom,\ altitude}$ (

n	a
Everest	8848
Manaslu	8163
Hidden Peak	8086

) =

nom	altitude
Everest	8848
Manaslu	8163
Hidden Peak	8086

Union, intersection et différence (1)

- Soit p et q deux expressions telles que :
  - $schéma(p) = schéma(q) = (A_1: D_1, \ldots, A_n: D_n)$
- Les opérateurs d'**union** ($\cup$), d'**intersection** ($\cap$) et de **différence** ($-$) sont définis par :
  - $schéma(p \cup q) = (A_1: D_1, \ldots, A_n: D_n)$
  - $ext(p \cup q) = \{t \mid t \in ext(p) \vee t \in \text{ext}(q)\}$
  - $schéma(p \cap q) = (A_1: D_1, \ldots, A_n: D_n)$
  - $ext(p \cap q) = \{t \mid t \in ext(p) \wedge t \in \text{ext}(q)\}$
  - $schéma(p - q) = (A_1: D_1, \ldots, A_n: D_n)$
  - $ext(p - q) = \{t \mid t \in ext(p) \wedge t \notin ext(q)\}$

Union, intersection et différence (2)

n	a
K2	8611
Everest	8848
Manaslu	8163

−

n	a
Everest	8848
Manaslu	8163

=

s	a
K2	8611

Produit cartésien (1)

- Soit p et q deux expressions telles que :
  - $schéma(p) = (A_1, \ldots, A_m)$
  - $schéma(q) = (B_1, \ldots, B_n)$
  - $\{A_1, \ldots, A_m\} \cap \{B_1, \ldots, B_n\} = \varnothing$
- Le **produit cartésien** $(\times)$ est défini par :
  - $schéma(p \times q) = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $ext(p \times q) =$

    $\{u \text{ conc } v \mid u \in ext(p) \wedge v \in ext(q)\}$

Produit cartésien (2)

n	**a**
Everest	8848

×

ns	**p**
Everest	Népal
Everest	Chine

=

n	**a**	**ns**	**p**
Everest	8848	Everest	Népal
Everest	8848	Everest	Chine

Sélection (1)

- L'opérateur de **sélection** (σ) extrait l'ensemble des *n*-uplets d'une relation qui vérifient une condition donnée.
- Une condition est une expression booléenne (c.-à-d. dont la valeur est vraie ou fausse) :
  - dont les opérandes sont soit des valeurs de domaine, soit des noms d'attributs de la relation à laquelle s'applique la sélection ;
  - les opérateurs sont les comparateurs : <, <=, =, >=, >, != et les connecteurs logiques : $\wedge$, $\vee$, $\neg$.
- Par exemple :
  - `altitude > 8500 ∧ (face = "S" ∨ face = "SO")`

  est une condition applicable aux *n*-uplets de la relation sommet.

Sélection (2)

- Soit r une expression telle que :

  $schéma(r) = (A_1, ..., A_n)$

- L'opérateur σ est défini par :
  - $schéma(\sigma_{cond}) = (A_1, ..., A_n)$
  - $ext(\sigma_{cond}) = \{t \mid t \in ext(r) \wedge val(cond)\}$

Sélection (3)

$\sigma_{a > 8500}$ (

n	a
Everest	8848
Manaslu	8163
Hidden Peak	8086

) =

n	a
Everest	8848

Projection (1)

- L'opérateur de **projection** (Π) extrait l'ensemble des valeurs d'un constituant d'une relation.
- Soit r une expression telle que :

  $schéma(r) = (A_1 : D_1, \ldots, A_n : D_n)$
- L'opérateur de projection est défini par :
  - $schéma(\Pi_{A_1, \ldots, A_n}(r)) = (A_1 : D_1, \ldots, A_n : D_n)$
  - $ext(\Pi_{A_1, \ldots, A_n}(r)) =$

    $\{(t.A_1, \ldots, t.A_n) \mid t \in ext(r)\}$

Projection (2)

$\pi_{ns,\ f}$(

p	ns	f
Népal	Everest	SE
Chine	Everest	SE
Népal	Manaslu	NE
Chine	Cho Oyu	O
Népal	Cho Oyu	O
Pakistan	Nanga Parbat	N

) =

ns	f
Everest	SE
Manaslu	NE
Cho Oyu	O
Nanga Parbat	N

Jointure

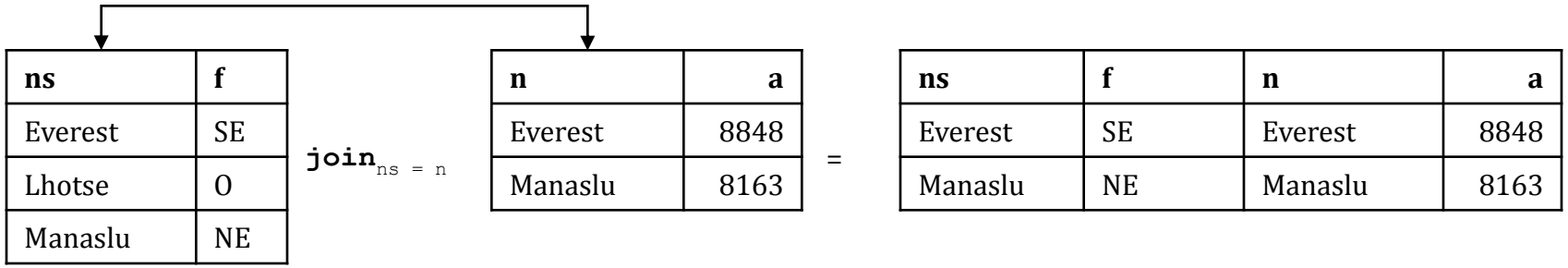
- La **jointure** est l'opération qui permet de concaténer des *n*-uplets provenant de deux tables et vérifiant une certaine condition.
- On distingue trois types de jointure :
  - la **jointure interne**,
  - la **jointure externe**,
  - la **jointure naturelle**.
- La jointure n'est pas un opérateur primitif :
  - Elle peut être réalisée, selon son type, en combinant les opérateurs de produit cartésien, de sélection, de projection et de renommage.

Jointure interne (1)

- L'opérateur de **jointure interne** (`join`) fusionne deux relations en produisant toutes les concaténations possibles de leurs n-uplets respectifs qui vérifient une certaine condition.
- La jointure interne est équivalente à un produit cartésien suivi d'une sélection :

$$p \ \texttt{join}_{cond} \ q \equiv \sigma_{cond}(p \times q)$$

Jointure interne (2)



ns	f
Everest	SE
Lhotse	O
Manaslu	NE

$\mathbf{join}_{ns = n}$

n	a
Everest	8848
Manaslu	8163

=

ns	f	n	a
Everest	SE	Everest	8848
Manaslu	NE	Manaslu	8163

Jointure externe (1)

- L'opérateur de **jointure externe** permet d'ajouter au résultat d'une jointure interne, les *n*-uplets de l'une, de l'autre ou des deux relations qui ne peuvent pas être joints.
- Soit *P* et *Q* les relations à joindre, on distingue :
  - la **jointure externe totale** (`ejoin`) qui ajoute les *n*-uplets de *P* et de *Q* qui ne peuvent pas être joints,
  - la **jointure externe gauche** (`lejoin`) qui n'ajoute que les n-uplets de *P* qui ne se joignent pas avec un *n*-uplet de *Q*,
  - la **jointure externe droite** (`rejoin`) qui n'ajoute que les n-uplets de *Q* qui ne se joignent pas avec un *n*-uplet de *P*.
- Les *n*-uplets ainsi rajoutés sont complétés par des valeurs nulles pour les attributs qu'ils ne possèdent pas.

Jointure externe (2)

ns	f
Everest	SE
Lhotse	O

$\texttt{ejoin}_{ns = n}$

n	a
Everest	8848
Manaslu	8163

=

ns	f	n	a
Everest	SE	Everest	8848
Lhotse	O	_	_
_	_	Manaslu	8163

ns	f
Everest	SE
Lhotse	O

$\texttt{lejoin}_{ns = n}$

n	a
Everest	8848
Manaslu	8163

=

ns	f	n	a
Everest	SE	Everest	8848
Lhotse	O	_	_

ns	f
Everest	SE
Lhotse	O

$\texttt{rejoin}_{ns = n}$

n	a
Everest	8848
Manaslu	8163

=

ns	f	n	a
Everest	SE	Everest	8848
_	_	Manaslu	8163

Jointure externe (3)

- Soit p et q deux expressions telles que :
  - $schéma(p) = (A_1, ..., A_m)$
  - $schéma(q) = (B_1, ..., B_n)$
  - $\{A_1, ..., A_m\} \cap \{B_1, ..., B_n\} = \varnothing$
- L'opérateur `ejoin` est défini par :
  - $schéma(p \ \texttt{ejoin}_{cond} \ q) = (A_1, ..., A_m, B_1, ..., B_n)$
  - $ext(p \ \texttt{ejoin}_{cond} \ q) = p_{ext} \cup j \cup q_{ext}$

    où :
    - $j = p \ \texttt{join}_{cond} \ q$
    - $p_{ext} = (p - \pi_{A_1, ..., A_m}(j)) \times \{(\texttt{nul}, ..., \texttt{nul})\}$
    - $q_{ext} = \{(\texttt{nul}, ..., \texttt{nul})\} \times (q - \pi_{B_1, ..., B_n}(j))$

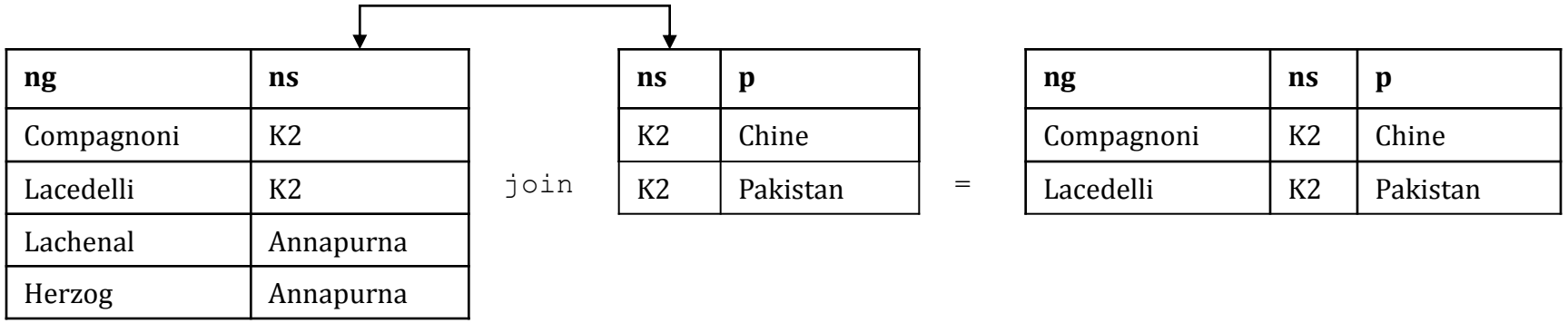
n-uplet

m-uplet

Jointure naturelle (1)

- Dans une jointure qu'elle soit interne ou externe, on peut omettre la condition lorsqu'elle se restreint à ce que les attributs de même nom de chacune des relations à joindre, doivent avoir la même valeur :
  - on appelle **jointure naturelle** ce cas particulier de jointure.
- La jointure naturelle sera notée en omettant la condition sous l'opérateur de jointure considéré.
- Dans la relation produite, les attributs communs aux deux relations n'apparaîtront qu'une seule fois.
- Si les deux relations à joindre n'ont pas d'attributs communs, alors la jointure naturelle est équivalente au produit cartésien.

Jointure naturelle (2)



ng	ns
Compagnoni	K2
Lacedelli	K2
Lachenal	Annapurna
Herzog	Annapurna

join

ns	p
K2	Chine
K2	Pakistan

=

ng	ns	p
Compagnoni	K2	Chine
Lacedelli	K2	Pakistan

Jointure naturelle (3)

- Soit p et q deux expressions telles que :
  - $schéma(p) = (A_1, ..., A_m)$
  - $schéma(q) = (B_1, ..., B_n)$
  - $\{A_1, ..., A_m\} \cap \{ B_1, ..., B_n\} = \{C_1, ..., C_k\}$
- La jointure naturelle est définie par :
  - $schéma(p \ \ j \ \ q) = (A_1, ..., A_m, L)$
  - $p \ \ j \ \ q = \pi_{A1, ..., Am, L}(p \ \ \texttt{join}_{C_1 = C'_1 \wedge ... \wedge C_k = C'_k} \ \ q')$

  où :
  - j est l'un des opérateurs `join`, `ejoin`, `lejoin`, `rejoin`,
  - L est la liste des attributs de q dans laquelle les attributs $C_1, ..., C_k$ ont été supprimés.
  - q' est la relation obtenue à partir de q en renommant respectivement $C'_1, ..., C'_k$ les attributs $C_1, ..., C_k$ avec $\{A_1, ..., A_m, B_1, ..., B_n\} \cap \{C'_1, ..., C'_k\} = \varnothing$

Requêtes en algèbre relationnelle (1)

- *Nom et altitude des sommets de plus de 8000 m dont la 1ère ascension a été réalisée par la face nord ?*
  - réponse := $\pi_{nom,\ altitude}(\sigma_{face\ =\ "N"}(sommet))$;
- *Nom et altitude des sommets de plus de 8000 m dont la 1ère ascension a été réalisée par Hermann Buhl ?*
  - ahb := $\sigma_{prénom\ =\ "Hermann"\ \wedge\ nom\ =\ "Buhl"}(ascension)$
  - réponse := $\pi_{nom,\ altitude}(sommet\ join_{nom\ =\ nom\_sommet}\ ahb)$
- *Nom des sommets de plus de 8000 m dont l'altitude est supérieure à celle du Makalu ?*
  - am := $\rho_{altitude\_makalu}(\pi_{altitude}(\sigma_{nom\ =\ "Makalu"}(sommet)))$;
  - réponse := $\pi_{nom}(sommet\ join_{altitude\ >\ altitude\_makalu}\ am)$;

Requêtes en algèbre relationnelle (2)

- *Prénom et nom des grimpeurs ayant réalisé la 1ère ascension d'un sommet sommets de plus de 8000 m du Népal ?*
  - asn := ascension join $\sigma_{pays = "Népal"}$(localisation);
    réponse := $\pi_{prénom\_grimpeur,\ nom\_grimpeur}$(asn);

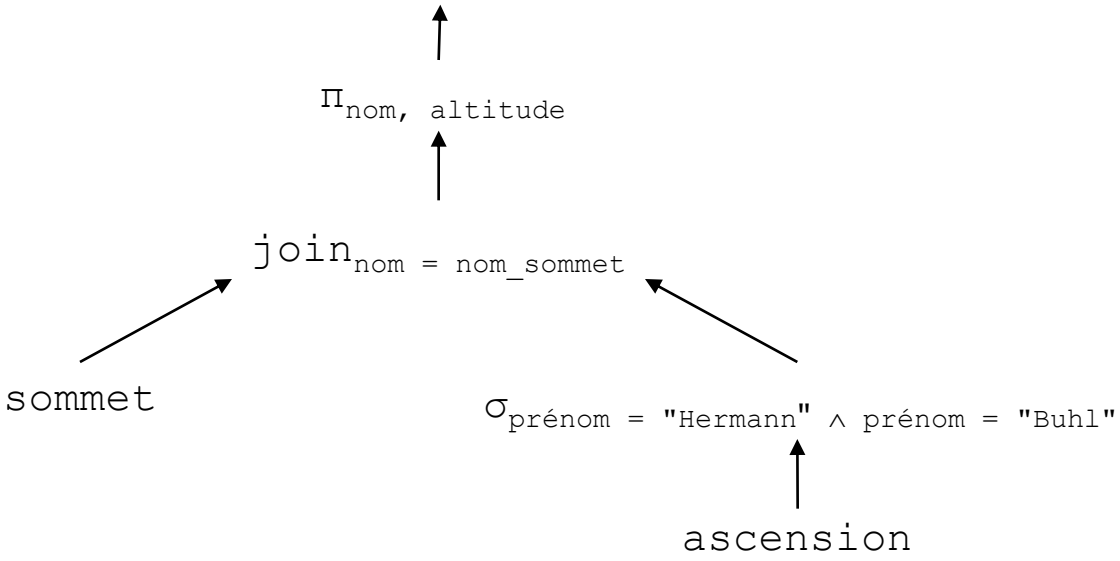
- *Nom des sommets de plus de 8000 m qui ne sont pas au Népal ?*
  (on sélectionne les noms des sommets qui ne joignent pas avec les sommets du Népal)
  - nsn := π_{nom_sommet}($\sigma_{pays = "Népal"}$(localisation))
    réponse := π_{nom}($\sigma_{nom\_sommet = nul}$(sommet ejoin$_{nom = nom_sommet}$ nsn));

- *Nom des sommets de plus de 8500 mètres situés au Népal mais pas sur la frontière chinoise ?*
  - ns := ρ_{nom_sommet}(π_{nom}($\sigma_{altitude > 8500}$(sommet)));
    nsn := π_{nom_sommet}($\sigma_{pays = "Népal"}$(localisation));
    nsc := π_{nom_sommet}($\sigma_{pays = "Chine"}$(localisation));
    réponse := ns ∩ nsn ∩ nsc;

Arbre de requête (1)

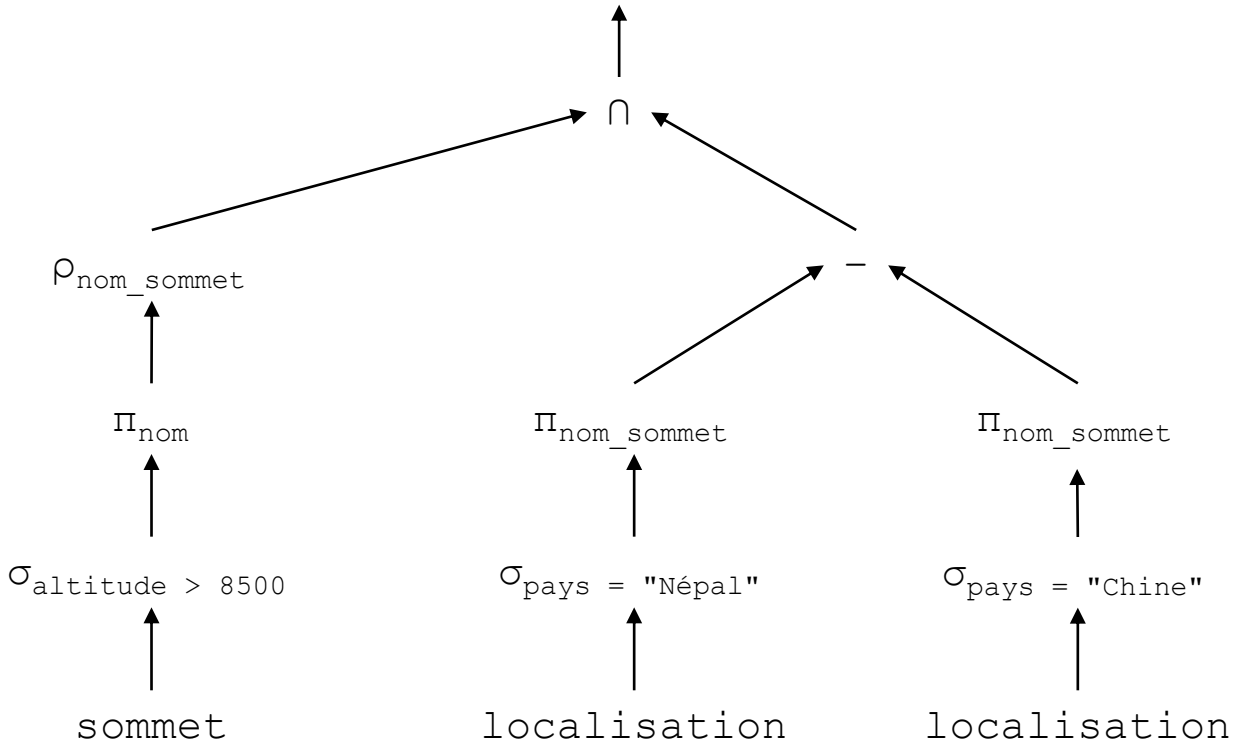
- Une requête de l'algèbre relationnelle peut être représentée par un arbre, appelé **arbre de requête**, dont les nœuds sont des opérateurs et les arcs des extensions de relations.
- On considérera un nom de relation comme l'opérateur qui produit l'extension de la relation ayant ce nom.

Arbre de requête (2)



Nom et altitude des sommets dont la 1ère ascension a été réalisée par Hermann Buhl ?

Arbre de requête (3)



Nom des sommets de plus de 8500 mètres situés au Népal mais pas sur la frontière chinoise ?

SQL

Introduction

- Le langage **SQL** est la version commercialisée du langage SEQUEL du prototype System R développé à IBM San José à partir de 1975.
- Il a depuis fait l'objet de plusieurs normes dont la plus récente est SQL-1992, qui est implantée plus ou moins complètement par tous les SGBD relationnels actuellement disponibles.
- Une nouvelle norme SQL:1999 a vu le jour, qui concerne les SGBD objets-relationnels.
- Le langage décrit dans ce chapitre est conforme à la norme SQL-1992.

Objectifs

- SQL est un langage qui permet de manipuler (c.-à-d. créer, modifier, supprimer, interroger) tous les types d'objets d'une BD relationnelle :
  - la BD,
  - les domaines,
  - les tables,
  - les règles,
  - les vues,
  - les index…

Panorama des commandes SQL

- Sur les objets :
  - `CREATE` *t n*
    - pour créer un objet de type *t* et de nom *n*
  - `ALTER` *t n*
    - pour modifier le schéma d'un objet de type *t* et de nom *n*
  - `DROP` *t n*
    - pour supprimer un objet de type *t* et de nom *n*
- Sur les tables :
  - `INSERT`
    - pour ajouter des lignes à une table
  - `REPLACE`
    - pour modifier des lignes d'une table
  - `DELETE`
    - pour supprimer des lignes d'une table
  - `SELECT`
    - pour extraire des données à partir de tables existantes

Principaux types de valeur

Catégorie	Type	Instance	Constante littérales (exemples)
Booléens	`BOOLEAN`	*vrai* ou *faux*	`TRUE FALSE`
Numériques	`INT` ou `INTEGER`	nombre entier	`1515 -269`
	`REAL` ou `FLOAT`	nombre flottant	`273.15 2731.5E-1`
	`DECIMAL(`*n*`, `*d*`)`	nombre décimal à *n* chiffres à *d* chiffres après la virgule	`3.14` est une instance du type `DECIMAL (3, 2)`
Textuels	`CHAR(`*n*`)`	chaîne de caractères de longueur fixe *n*	`'BD'` (pourra être complété par *n* – 2 blancs si *n* > 2)
	`VARCHAR(`*n*`)`	chaîne de caractères de longueur maximum *n*	`'bonjour'`
Temporels	`DATE`	date : jour, mois et année,	`DATE '2007-14-02'`
	`TIME`	temps : heure, minutes, secondes	`TIME '13:15:00'`
	`TIMESTAMP`	date + temps	`TIMESTAMP '2007-14-02 13:15:00'`
	`INTERVAL`	durée en années, mois, jours, heures, minutes ou secondes	`INTERVAL '6 days 4 hours'`

Création de domaine

```
CREATE DOMAIN orientation
  CHAR(2)
  CHECK (VALUE IN ('N', 'S', 'O', 'E',
                   'NO', 'SO','NE', 'SE'));
CREATE DOMAIN nom_pays_himalayen
  VARCHAR(20)
  CHECK (VALUE IN ('Chine', 'Inde', 'Népal',
                   'Pakistan'));
CREATE DOMAIN nom_pays
  VARCHAR(20)
  CHECK (VALUE IN ('Afghanistan', 'Afrique du Sud',
                   'Albanie', …));
```

Création de table (1)

- `CREATE TABLE nom (`

  nom_attribut type valeur_par_défaut contrainte

  …

  nom_attribut type valeur_par_défaut contrainte

  contrainte de table

  …

  contrainte de table)

Création de table (2)

```
CREATE TABLE sommet (
 nom VARCHAR(20) PRIMARY KEY,
 altitude INTEGER CHECK (altitude >= 8000),
 date DATE,
 face orientation)
```

Si l'on avait voulu que l'altitude soit exprimée en mètres et centimètres, on aurait pu définir l'attribut altitude par :

- `altitude DECIMAL(6, 2)`

```
CREATE TABLE localisation (
 nom_sommet VARCHAR(20) REFERENCES sommet(nom),
 pays nom_pays_himalayen,
 PRIMARY KEY (nom_sommet, pays));
```

Création de table (3)

```
CREATE TABLE grimpeur (
 nom VARCHAR(20),
 prénom VARCHAR(20),
 pays nom_pays,
 PRIMARY KEY (nom, prénom))
```

```
CREATE TABLE ascension (
 nom_grimpeur VARCHAR(20),
 prénom_grimpeur VARCHAR(20),
 nom_sommet VARCHAR(20) REFERENCES sommet(nom),
 PRIMARY KEY (nom_grimpeur,
              prénom_grimpeur,
              nom_sommet),
 FOREIGN KEY (nom_grimpeur, prénom_grimpeur)
  REFERENCES grimpeur (nom, prénom));
```

Structure d'une requête SQL

- Une requête SQL est composée à partir :
  - d'un opérateur de base :

    ```
    SELECT
    FROM
    WHERE
    GROUP BY
    HAVING
    ORDER BY
    ```

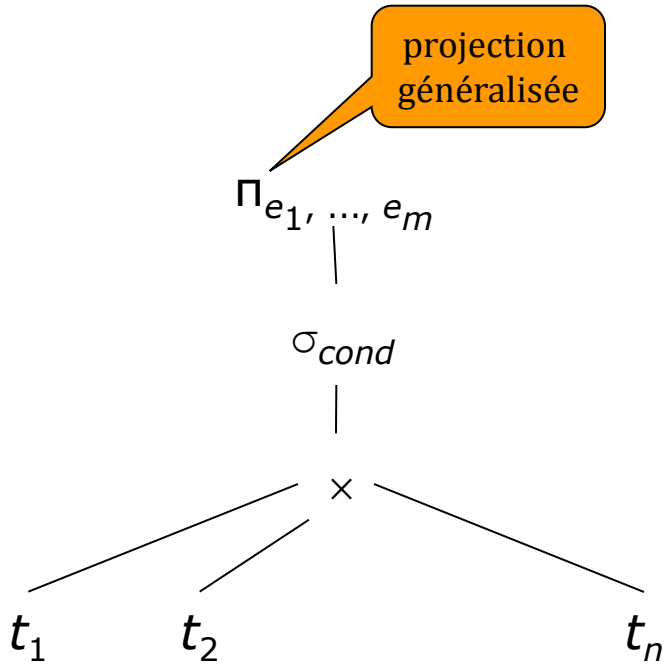
    qui produit une table ;
  - d'opérateurs d'union, d'intersection ou de différence de tables ;
  - des opérateurs de jointures.

L'opérateur SELECT…FROM…WHERE

- constante littérale
- nom d'attribut éventuellement qualifié par le nom de la table auquel il appartient
- opération construite à l'aide des opérateurs SQL
- requête SQL produisant une table monocase

SELECT e_1, …e_m
FROM t_1, …, t_n
WHERE *cond*

- nom de table de la BD
- opération de jointure interne ou externe, naturelle ou non
- requête SQL



Requêtes monotable (1)

- *Q1. Noms des sommets de plus 8000 m et année de leur 1ère ascension ?*
  - ```
    SELECT nom
    FROM sommet;
    ```

- *Q2a. Sommets de plus de 8500 m dont la 1ère ascension a été réalisée après 1955 sur une face sud ou sud-est ?*
  - ```
    SELECT *
    FROM sommet
    WHERE altitude > 8500 AND
          (face = 'SE' OR face = 'SO');
    ```

- *Q2b. Idem avec l'opérateur `IN`*
  - ```
    SELECT *
    FROM sommet
    WHERE altitude > 8500 AND face IN ('S', 'N');
    ```
    ```

Requêtes monotable (2)

- *Q3. Nom et année de la 1ère ascension de chaque sommet de plus de 8000 m ?*
  - ```
    SELECT nom, EXTRACT('year' FROM date) AS année
    FROM sommet;
    ```

- *Q4. Nom et altitude en pieds des sommets de plus 8000 m dont le nom contient le mot « Peak » ?*
  - ```
    SELECT nom, 3.29 * altitude AS altitude_en_pieds
    FROM sommet
    WHERE nom LIKE '%Peak%';
    ```

- *Q5. Nom et altitude des sommets dont l'altitude est comprise entre 8100 m et 8500 m ?*
  - ```
    SELECT nom, altitude
    FROM sommet
    WHERE altitude BETWEEN 8100 AND 8500;
    ```
    ```

Jointure et utilisation de synonymes

- *Q6a. Donner pour chaque 1ère ascension d'un sommet de plus de 8500 m le nom du grimpeur, le nom du sommet et son altitude ?*
  - ```
    SELECT ascension.nom_grimpeur, sommet.nom, sommet.altitude
    FROM ascension, sommet
    WHERE ascension.nom_sommet = sommet.nom AND
          sommet.altitude > 8500;
    ```

- *Q6b. Idem en désignant les tables* `ascension` *et* `sommet` *par les synonymes* `a` *et* `s` *?*
  - ```
    SELECT a.nom_grimpeur, s.nom, s.altitude
    FROM ascension a, sommet s
    WHERE a.nom_sommet = s.nom AND s.altitude > 8500;
    ```

- *Q6c. Idem sans qualifier les attributs car il n'y a pas d'ambiguïté sur leur table de provenance ?*
  - ```
    SELECT nom_grimpeur, nom, altitude
    FROM ascension, sommet
    WHERE nom_sommet = nom AND altitude > 8500;
    ```
    ```

Éliminer les doubles : la clause DISTINCT

- *Q7. Nom des pays dont un grimpeur a réalisé la 1ère ascension d'un sommet de plus 8000 m ?*
  - ```
    SELECT DISTINCT pays
    FROM grimpeur;
    ```

- *Q8. Nom et prénom des grimpeurs ayant réalisé la 1ère ascension d'un sommet de plus de 8000 m du Pakistan ?*
  - ```
    SELECT DISTINCT a.nom_grimpeur,
    a.prénom_grimpeur
    FROM ascension a, localisation l
    WHERE a.nom_sommet = l.nom_sommet AND
          l.pays = 'Pakistan';
    ```

Opérateurs d'agrégation

- *Q9. Nombre de sommets de plus de 8000 m ?*
  - ```
    SELECT COUNT(*)
    FROM sommet;
    ```
- *Q10. Nombre de pays possédant un sommet de plus de 8000 m ?*
  - ```
    SELECT COUNT(DISTINCT pays)
    FROM localisation;
    ```
- *Q11. Altitudes minimale, moyenne et maximale des sommets de plus de 8000 m ?*
  - ```
    SELECT MIN(altitude), AVG(altitude), MAX(altitude)
    FROM sommet;
    ```
    ```

Sous-requêtes : comparaison valeur/table

- *Q12. Nom des sommets dont l'altitude est supérieure à l'altitude du Makalu ?*
  - ```
    SELECT nom
    FROM sommet
    WHERE altitude >
      (SELECT altitude
       FROM sommet
       WHERE nom = 'Makalu');
    ```
    ```

Sous-requêtes : IN et NOT IN

- *Q13. Nom des pays dont un grimpeur a réalisé la 1ère ascension du Dhaulagiri ?*
  - ```
    SELECT DISTINCT pays
    FROM grimpeur
    WHERE (nom, prénom) IN
     (SELECT nom_grimpeur, prénom_grimpeur
      FROM ascension
      WHERE nom_sommet = 'Dhaulagiri');
    ```

- *Q14. Nom et prénom des grimpeurs n'appartenant pas à un pays possédant un sommet de plus 8000 m ?*
  - ```
    SELECT nom, prénom
    FROM grimpeur
    WHERE pays NOT IN
     (SELECT pays
      FROM localisation);
    ```

Sous-requêtes : opérateur θALL

- *Q15. Nom du sommet le plus haut ?*
  - ```
    SELECT nom
    FROM sommet
    WHERE altitude >=ALL
      (SELECT altitude
       FROM sommet);
    ```
    ```

Sous-requêtes : quantification existentielle

- *Q16. Nom des grimpeurs ayant réalisé une 1ère ascension d'un sommet de plus de 8000 m avec Hermann Buhl ?*
  - ```
    SELECT DISTINCT a1.nom_grimpeur,
                    a1.prénom_grimpeur
    FROM ascension a1
    WHERE EXISTS
     (SELECT *
      FROM ascension a2
      WHERE a2.nom_grimpeur = 'Buhl' AND
            a2.prénom_grimpeur = 'Hermann' AND
            a2.nom_grimpeur <> a1.nom_grimpeur AND
            a2.prénom_grimpeur <>a1.prénom_grimpeur AND
            a2.nom_sommet = a1.nom_sommet);
    ```
    ```

Union, intersection et différence

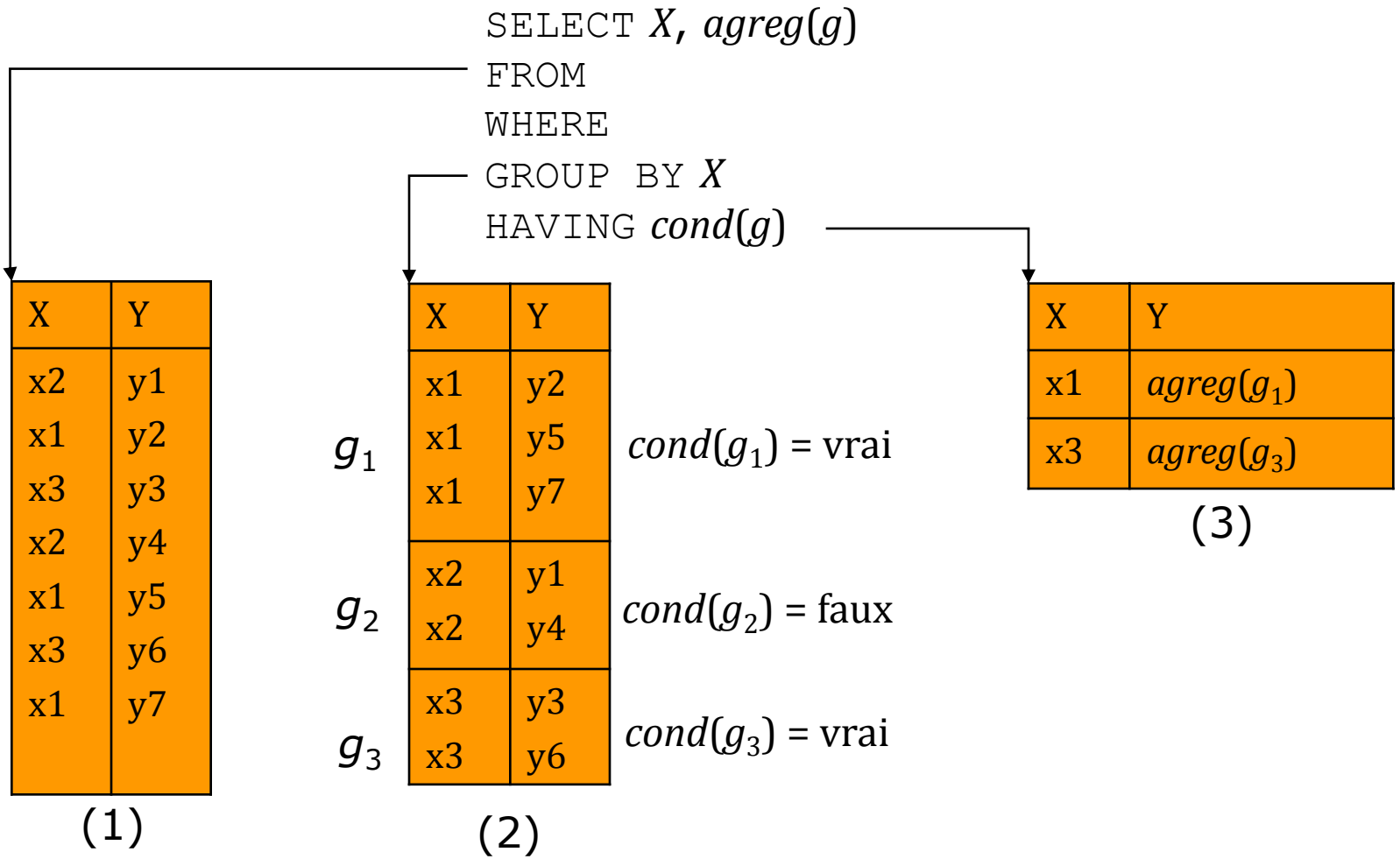
- *Q17. Nom des sommets de plus de 8500 m situés au Népal mais pas sur la frontière avec la Chine ?*
  - ```
    (SELECT nom
     FROM sommet
     WHERE altitude > 8500)
    INTERSECT
    (SELECT nom_sommet
     FROM localisation
     WHERE pays = 'Népal')
    EXCEPT
     (SELECT nom_sommet
      FROM localisation
      WHERE pays = 'Chine');
    ```
    ```

Tri

- *Q18. Lister les noms des sommets de plus de 8000 m par ordre alphabétique.*
  - ```
    SELECT nom
    FROM sommet
    ORDER BY nom ASC;
    ```

- *Q19. Lister les triplets (p, s, a) où p est un nom de pays, s est le nom d'un sommet de plus de 8000 m de p et a est l'altitude de s. Trier le résultat par ordre alphabétique de pays et par ordre décroissant d'altitude.*
  - ```
    SELECT l.pays, s.nom, s.altitude
    FROM localisation l, sommet s
    WHERE l.nom_sommet = s.nom
    ORDER BY l.pays ASC, s.altitude DESC;
    ```

Groupement et agrégation (1)



Groupement et agrégation (2)

- *Q20. Nombre de sommets de plus de 8000 m de chaque pays en possédant ?*
  - ```
    SELECT pays, COUNT(*)
    FROM localisation
    GROUP BY pays;
    ```

- *Q21. Donner pour chaque pays : son nom et le nombre de sommets de plus de 8000 m dont la 1ère ascension a été réalisée par un grimpeur de ce pays. Trier le résultat par nombre décroissant de sommets ?*
  - ```
    SELECT g.pays, COUNT(DISTINCT a.nom_sommet)
    FROM ascension a, grimpeur g
    WHERE a.nom_grimpeur = g.nom
    GROUP BY g.pays
    ORDER BY 2 DESC;
    ```

Groupement et agrégation (3)

- *Q22. Noms des sommets de plus de 8000 m dont la 1ère ascension a été réalisé par plus de 5 grimpeurs et nombre de ces grimpeurs ?*
  - ```
    SELECT nom_sommet, COUNT(*)
    FROM ascension
    GROUP BY nom_sommet
    HAVING COUNT(*) > 5;
    ```

- *Q23. Noms et prénom des grimpeurs ayant réalisé la 1ère ascension de plus d'un sommet de plus de 8000 m ?*
  - ```
    SELECT g.nom, g.prénom
    FROM ascension a, grimpeur g
    WHERE a.nom_grimpeur = g.nom AND
    a.prénom_grimpeur = g.prénom
    GROUP BY g.nom, g.prénom
    HAVING COUNT(*) > 1;
    ```

Construction de tables dans la clause FROM

- *Q25. Nombre maximum de sommets de plus 8000 m localisés dans un même pays ?*
  - ```
    SELECT max(histo.nb)
    FROM (SELECT pays, COUNT(*) AS nb
          FROM localisation
          GROUP BY pays) AS histo;
    ```
    ```

Opérateurs de jointure dans la clause FROM

- *Q26. Donner le prénom, le nom et le nom du sommet pour chaque grimpeur ayant réalisé la 1ère ascension d'un sommet de plus de 8000 m de l'Inde ?*
  - ```
    SELECT prénom_grimpeur, nom_grimpeur, nom_sommet
    FROM ascension NATURAL INNER JOIN localisation
    WHERE pays = 'Inde';
    ```

- *Q27. Parmi tous les grimpeurs ayant réalisé la 1ère ascension d'un sommet de plus de 8000 m, donner le pourcentage de ceux qui l'ont fait sur un sommet du Népal ?*
  - ```
    SELECT (CAST(COUNT(pays) AS FLOAT) / COUNT(*)) * 100
    FROM (SELECT *
          FROM ascension) AS a
         NATURAL LEFT OUTER JOIN
         (SELECT *
          FROM localisation
          WHERE pays = 'Népal') AS l;
    ```

Manipulation des valeurs nulles (1)

- Les comparateurs `IS NULL` et `IS NOT NULL` permettent de tester si une valeur est nulle :
  - *e* `IS NULL` a la valeur `TRUE` si *e* a la valeur `NULL`, elle a la valeur `FALSE` sinon,
  - *e* `IS NOT NULL` a la valeur Vrai si *e* a la valeur `NULL`, elle a la valeur `FALSE` sinon.
- Si l'un des opérandes d'un opérateur a la valeur `NULL`, le résultat de l'opération est la valeur `NULL`.
- Pour traiter les comparateurs (autres que `IS NULL` et `IS NOT NULL`) et les connecteurs logiques, on introduit une troisième valeur de vérité : `UNKNOWN`, en plus des valeurs `TRUE` et `FALSE`.
  - La comparaison entre la valeur `NULL` et toute autre valeur y compris `NULL` a pour valeur `UNKNOWN`.
  - On redéfinit les tables de vérité des trois connecteurs logiques en tenant compte de cette 3^{e} valeur de vérité.

Manipulation des valeurs nulles (2)

Tables de vérité à 3 valeurs des connecteurs logiques

AND	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

OR	T	F	U
T	T	F	T
F	F	F	U
U	T	U	U

NOT	
T	F
F	T
U	U

Manipulation des valeurs nulles (3)

- Supposons que la relation sommet ait l'extension suivante :

sommet			
nom	**altitude**	**année**	**face**
Everest	8848	_	N
K2	8611	1954	SE

- La requête :
  - `SELECT COUNT(*)`
    `FROM sommet`
    `WHERE année < 1950`

  retourne le *n*-uplet `(0)`.
- La requête :
  - `SELECT nom`
    `FROM sommet`
    `WHERE année IS NULL;`

  retourne le *n*-uplet `('Everest')`.
- La requête :
  - `SELECT nom`
  - `FROM sommet`
  - `WHERE année IS NOT NULL;`

  retourne le *n*-uplet `('K2')`.

Manipulation de valeurs nulles (4)

- *Q28. Noms des sommets de plus de 8000m qui ne sont pas situés au Népal ?*
  - ```
    SELECT s.nom
    FROM sommet AS s
         LEFT OUTER JOIN
         (SELECT DISTINCT nom_sommet
          FROM localisation
          WHERE pays = 'Népal') AS l
         ON s.nom = l.nom_sommet
    WHERE l.nom_sommet IS NULL;
    ```

  (on sélectionne les noms des sommets qui ne joignent pas avec les sommets du Népal)
    ```

Vues (1)

- Une **vue** est une relation virtuelle, exprimée en termes d'une requête SQL.
- Les vues permettent :
  - de simplifier l'expression des requêtes,
  - de définir des sous-ensembles de la BD sur lesquels, il sera possible de spécifier des droits d'accès.

Vues (2)

- *Créer une vue* `ascension_népalaise` *contenant le nom des sommets de plus de 8000 m dont la 1ère ascension a été réalisée par un grimpeur népalais et l'année de cette ascension.*
  - ```
    CREATE VIEW ascension_népalaise(sommet, année) AS
    SELECT DISTINCT a.nom_sommet, s.année
    FROM ascension a, grimpeur g, sommet s
    WHERE a.nom_grimpeur = g.nom AND
          a.prénom_grimpeur = g.prénom AND
          a.nom_sommet = s.nom AND
          g.pays = 'Népal';
    ```
- *Nombre de sommets de plus de 8000 m dont la 1ère ascension a été réalisée par un grimpeur népalais ?*
  - ```
    SELECT COUNT(*) FROM ascension_népalaise;
    ```
- *Supprimer la vue* `ascension_népalaise`.
  - ```
    DROP VIEW ascension_népalaise;
    ```
    ```

Utilisation d'une vue pour décomposer une requête complexe

- *Nom des pays possédant le plus de sommets de plus de 8000m ?*
  - ```
    CREATE VIEW histo AS
    SELECT pays, COUNT(*) AS nb
    FROM localisation
    GROUP BY pays;
    ```
  - ```
    SELECT pays
    FROM histo
    WHERE nb =
      (SELECT MAX(nb)
       FROM histo);
    ```
  - ```
    DROP VIEW histo;
    ```
    ```

Mise à jour d'une table (1)

- *Insérer le sommet Everest ?*
  - ```
    INSERT INTO sommet
    VALUES ('Everest ', 8850, 1953, 'SE')
    ```

- *Mettre à jour l'altitude de l'Everest ?*
  - ```
    UPDATE sommet
    SET altitude = 8848
    WHERE nom = 'Everest';
    ```

Mise à jour d'une table (2)

- *Renommer* `altitude_en_mètres` *la colonne* `altitude`*, ajouter une colonne* `altitude_en_pieds` *et y insérer les altitudes en pieds des sommets calculées à partir des altitudes en mètres.*
  - ```
    ALTER TABLE sommet
    RENAME altitude TO altitude_en_mètres;
    ```
  - ```
    ALTER TABLE sommet
    ADD COLUMN altitude_en_pieds INTEGER;
    ```
  - ```
    UPDATE sommet
    SET altitude_en_pieds = 3.28 *
    altitude_en_mètres;
    ```
    ```

Mise à jour au travers d'une vue

- Pour être utilisée pour une mise à jour une vue doit vérifier les conditions suivantes :
  - La vue ne doit pas inclure la clause `DISTINCT`.
  - Chaque élément de la clause `SELECT` doit être un nom de colonne.
  - La clause `FROM` ne doit contenir qu'une seule table, elle-même modifiable.
  - La clause `WHERE` ne doit pas contenir de sous-requête.
  - La vue ne doit contenir ni clause `GROUP BY`, ni clause `HAVING`.

Maintien de l'intégrité

- Il existe plusieurs façons d'exprimer les contraintes d'intégrité d'une BD :
  - définition du domaine ou du type d'un attribut,
  - condition sur les valeurs des attributs d'un *n*-uplet (clause `CHECK` de SQL),
  - définition de la clé primaire et des clés étrangères d'une relation,
  - assertions générales,
  - déclencheurs.
- Une **assertion** est une formule logique qui doit être vraie quelque soit l'extension de la BD.
- Un **déclencheur** est une règle, dite **active**, de la forme :
  - « événement - condition - action »
  - L'action est déclenchée à la suite de l'événement, si la condition est vérifiée.
  - Une action peut être une vérification ou une mise à jour.

Assertions (1)

- Une **assertion** est une formule logique qui doit être vraie quelque soit l'extension de la BD.
- Attention ! la vérification d'une assertions peut être une opération coûteuse, car elle doit être mise en œuvre après chaque mise à jour de la BD.
- Une assertion est créée par la commande SQL :
  - `CREATE ASSERTION` *nom* `CHECK (`*condition*`)`

Assertions (2)

- Soit la BD contenant les deux relations suivantes :
  - employé(nom_emp, nom_dept, salaire)
  - département(nom_dept, directeur, nb_emp)
- La contrainte d'intégrité : « Tout employé du département 'Recherche' doit avoir un salaire supérieur à 3000 € », s'exprime par l'assertion suivante :

```
CREATE ASSERTION CHECK
  (NOT EXISTS
   (SELECT *
    FROM employe
    WHERE salaire <= 3000 AND
          nom_dept =
           (SELECT nom_dept
            FROM departement
            WHERE nom_dept = 'Recherche')))
```

Déclencheurs (1)

- Un **déclencheur** est une règle, dite **active**, de la forme :
  - « événement-condition-action »
  - L'action est déclenchée à la suite de l'événement et si la condition est vérifiée.
  - Une action peut être une vérification ou une mise à jour.

Déclencheurs (2)

- Un déclencheur est activé par une requête de mise à jour.
- Pour définir un déclencheur, il faut :
  - spécifier l'événement qui déclenche l'action en indiquant le type de la mise à jour (`INSERT`, `UPDATE`, `DELETE`), le nom de la relation et éventuellement le nom des attributs mis à jour ;
  - indiquer si l'action est réalisée avant, après ou à la place de la mise à jour ;
  - donner un nom à l'ancien et au nouveau *n*-uplet (uniquement le nouveau en cas d'insertion et uniquement l'ancien en cas de suppression) ;
  - décrire la condition sous laquelle se déclenche l'événement sous la forme d'une expression SQL booléenne, c.-à-d. une expression pouvant être placée dans une clause `WHERE` ;
  - décrire l'action à réaliser sous la forme d'une procédure SQL ;
  - indiquer si l'action est réalisée pour chaque *n*-uplet mis à jour ou une seule fois pour la requête.

Déclencheurs (3)

- *Création d'un déclencheur qui signale les mises à jour de salaire qui sont des diminutions et les empêchent.*
  - ```
    CREATE TRIGGER diminution_salaire
    BEFORE UPDATE OF salaire ON employe
    REFERENCING OLD ROW AS a
                NEW ROW AS n
    FOR EACH ROW
    WHEN n.salaire < a.salaire
    ```
    Procédure SQL signalant que le nouveau salaire est trop bas et empêchant la mise à jour ;
    ```

Déclencheurs (4)

- *Création d'un déclencheur mettant automatiquement à jour le nombre d'employés d'un département à la suite de l'insertion d'un nouvel employé.*
  - ```
    CREATE TRIGGER incrémenter_nombre_employés
    AFTER INSERT ON employé
    REFERENCING NEW ROW AS e
    FOR EACH ROW
     SET nb_emp = nb_emp + 1
     WHERE num_dept = e.num_dept);
    ```
    ```

Confidentialité

- Une BD doit être protégée des accès malveillants.
- La solution adoptée classiquement consiste à n'**autoriser** un utilisateur à effectuer une **opération** sur un **objet** que s'il en a obtenu le **droit** (ou le **privilège**).
- L'identification des utilisateurs se fait en général par leur nom et par un mot de passe. Des procédés plus sophistiqués peuvent être utilisés comme les cartes à puce ou la reconnaissance des empreintes digitales ou rétiniennes.
- Dans un SGBD relationnel :
  - les objets à protéger sont les tables et les vues,
  - les opérations sont `SELECT`, `INSERT`, `UPDATE` ou `DELETE`.
- L'attribution des droits peut être :
  - centralisé : l'administrateur a tous les droits sur tous les objets de la BD et peut transmettre certains de ces droits à d'autres utilisateurs.
  - décentralisée : l'utilisateur qui crée un objet a tous les droits sur cet objet et peut les transmettre en totalité ou en partie à d'autres utilisateurs.

Protection par les vues

- Les vues jouent un rôle important pour la confidentialité en permettant de spécifier de façon très fine les données auxquels un utilisateur a le droit d'accéder.
- On pourra spécifier un accès à l'affectation des employés dans les départements en définissant la vue suivante :
  - ```
    CREATE VIEW affectation(nom_emp, nom_dept) AS
      SELECT e.nom_emp, d.nom_dept
      FROM employe AS e, departement AS d
      WHERE e.nom_dept = d.nom_dept;
    ```
- Un utilisateur dont les droits d'accès se limitent à la vue `affectation` ne pourra connaître ni le numéro et le salaire d'un employé, ni le numéro et le directeur d'un département.
    ```

GRANT et REVOKE

- Une utilisatrice ayant créé les relations `employe` et `departement` ainsi que la vue `affectation` pourra :
  - Transmettre à l'utilisateur Jean Dupont le droit de consulter les employés et de modifier le salaire d'un employé :

    ```
    GRANT SELECT, UPDATE(salaire)
    ON employe
    TO 'Jean Dupont'
    ```

  - Transmettre à tous les utilisateurs le droit de consulter les affectations des employés :

    ```
    GRANT SELECT ON affectation TO PUBLIC
    ```

  - Retirer à Jean Dupont le droit de modifier le salaire d'un employé :

    ```
    REVOKE UPDATE(salaire) ON employe TO 'Jean Dupont'
    ```

    (Notons que Jean Dupont possède encore le droit de consulter les employés.)

Intégration de SQL dans un programme (1)

- Les principes sont les suivants :
- Chaque instruction SQL est préfixée par `EXEC SQL`.
- Ces instructions peuvent contenir des variables du programme qui :
  - sont préfixées par :
  - doivent avoir un type compatible avec les valeurs d'attributs qui leur seront affectées.
- Un mécanisme de curseur permet de parcourir une table ligne par ligne et d'affecter les valeurs de chaque ligne à des variables du programme.
- Le programme doit posséder une variable numérique `SQLCODE` dont la valeur résume l'exécution, correcte ou incorrecte, d'une instruction SQL.

Intégration de SQL dans un programme (2)

- *Affecter à la variable* `salaire` *le salaire de l'employé dont le nom est donné dans la variable* `nom` *?*
  - ```
    EXEC SQL BEGIN DECLARE SECTION;
    char nom[20];
    int salaire;
    EXEC SQL END DECLARE SECTION;
    main()
     {
     scanf("%s", nom);
     EXEC SQL CONNECT TO ma_bd;
     EXEC SQL
      SELECT salaire INTO :salaire
      FROM employe
      WHERE nom_emp = :nom;
     printf("salaire = %d", salaire);
     EXEC SQL DISCONNECT;
     }
    ```
    ```

Intégration de SQL dans un programme (2)

- *Afficher le salaire en dollars de chaque employé du département « Recherche » ?*
  - ```
    #include <stdio.h>
    EXEC SQL BEGIN DECLARE SECTION;
    char nom_emp[20];
    int salaire;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL DECLARE employe CURSOR FOR
     SELECT nom_emp, salaire
     FROM employe
     WHERE nom_dept IN
            (SELECT nom_dept
             FROM departement
             WHERE nom_dept = 'Recherche');
    …
    ```
    ```

Intégration de SQL dans un programme (3)

```
…
main()
 {
 EXEC SQL CONNECT TO ma_bd;
 EXEC SQL OPEN employe;
 EXEC SQL WHENEVER NOT FOUND DO BREAK;
 while (1)
  {
  EXEC SQL FETCH employe INTO :nom_emp, :salaire;
  printf("salaire en dollars de %s = %.2f\n",
         nom, 1.3 * :salaire);
  }
 EXEC SQL DISCONNECT;
 }
```

Normalisation

Le problème

- Mélanger dans une même relation des informations relatives à plusieurs entités entraîne, en général des redondances d'information qui provoquent les anomalies suivantes:
  - anomalies d'**insertion**,
  - anomalies de **mise à jour**,
  - anomalies de **suppression**.

Exemple

isbn	**titre**	**éditeur**	**pays**
2-212-09283-0	Bases de données – objet et relationnel	Eyrolles	France
2-7117-8645-5	Fondements des bases de données	Vuibert	USA
0-201-70872-8	Databases and Transaction Processing	Addison Wesley	USA
2-212-09069-2	Internet/Intranet et bases de données	Eyrolles	France

- Cette relation qui décrit des livres et leurs éditeurs, contient des redondances provoquant les anomalies suivantes :
  - insertion : il n'est pas possible d'insérer un livre sans connaître son éditeur. Attribuer une valeur nulle aux attributs éditeur et pays violerait l'intégrité d'entité : un attribut appartenant à la clé, ne doit pas avoir de valeur nulle.
  - mise à jour : si un éditeur change de pays, il faut modifier ce pays pour chacun des livres qu'il a édités.
  - suppression : si l'unique livre publié par un éditeur est supprimé, l'information sur cet éditeur est perdue. Attribuer une valeur nulle aux attributs isbn et titre, violerait l'intégrité d'entité.

Solution

- La solution à ces problèmes consiste à **normaliser** la relation en cause en la décomposant en plusieurs relations.
- Cette décomposition s'appuie sur les **dépendances** qui existent entre les attributs de la relation initiale :
  - dépendances **fonctionnelles**,
  - dépendances **multivaluées**.
- Par exemple, la dépendance entre l'ISBN d'un livre et son titre ou bien entre le nom d'un éditeur et son pays.

Dépendances fonctionnelles

- Il y a une **dépendance fonctionnelle** entre un constituant X et un constituant Y d'une relation R si pour toute extension de R, à chaque valeur de X il correspond toujours la même valeur de Y.
- On dit que X **détermine fonctionnellement** Y et l'on note :
  - $X \rightarrow Y$
- Un constituant $\{A_1, \ldots, A_n\}$ apparaissant à gauche ou à droite du signe $\rightarrow$ sera noté $A_1 \ldots A_n$.

Dépendances fonctionnelles triviales

- Une dépendance fonctionnelle $X \rightarrow Y$ est dite **triviale** si Y est un sous-ensemble de X ou bien si Y est vide.
- Par exemple, les dépendances :
  - `isbn titre` $\rightarrow$ `pitre`
  - `éditeur pays` $\rightarrow \varnothing$

  sont triviales.

Propriétés des dépendances fonctionnelles (1)

- A partir d'un ensemble de dépendances fonctionnelles, on peut en déduire d'autres.
- Par exemple, à partir des dépendances :
  - `éditeur` → `ville`
  - `ville` → `pays`

  on peut déduire, par transitivité, la dépendance :
  - `éditeur` → `pays`

Propriétés des dépendances fonctionnelles (2)

- Étant donné un ensemble de dépendances fonctionnelles F construit sur l'ensemble R des attributs d'une relation, l'ensemble $F+$ (**fermeture** de F) de toutes les dépendances fonctionnelles logiquement impliquées par F peut être calculé à partir des trois règles suivantes (axiomes d'Armstrong) :
  - (**réflexivité**) si $Y \subseteq X \subseteq R$ alors $X \rightarrow Y$
  - (**augmentation**) si $X \rightarrow Y$ et $Z \subseteq R$ alors $X \cup Z \rightarrow Y \cup Z$
  - (**transitivité**) si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$
- Des axiomes d'Armstrong on peut déduire deux règles plus pratiques pour calculer $F+$:
  - (**union**) si $X \rightarrow Y$ et $X \rightarrow Z$ alors $X \rightarrow Y \cup Z$.
  - (**décomposition**) si $X \rightarrow Y \cup Z$ alors $X \rightarrow Y$ et $X \rightarrow Z$.

Dépendances fonctionnelles et clés (1)

- On peut définir une clé d'une relation comme un sous-ensemble de ses attributs qui détermine tous les autres.
- Un constituant X d'une relation $R(A_1, \ldots, A_n)$ est une clé de cette relation si et seulement si :
  - $X \rightarrow A_1, \ldots, A_n$,
  - il n'existe pas de constituant Y inclus dans X tel que $Y \rightarrow A_1, \ldots, A_n$.
- Etant données une relation et l'une de ses clés, on appelle :
  - **attribut clé** un attribut qui appartient à cette clé,
  - **attribut non clé**, un attribut qui n'y appartient pas.
- On appelle **super-clé** un constituant qui inclut une clé.

Dépendances fonctionnelles et clés (2)

- Par exemple dans la relation

  `personne(nom, prénom, age)`

  de clé `{nom, prénom}` :
  - `nom` et `prénom` sont des attributs clés,
  - `age` est un attribut non clé,
  - `{nom, prénom, age}` est une super-clé.

Normalisation d'une relation

- La **normalisation** d'une relation consiste à la décomposer en un ensemble de relations telles qu'aucune des relations obtenues ne possède les anomalies de redondance, de mise à jour et de suppression.
- Relativement aux dépendances fonctionnelles, on distingue 4 formes normales :
  - la 1ère forme normale,
  - la 2e forme normale,
  - la 3e forme normale,
  - la forme normale de Boyce-Codd qui est la plus aboutie.
- Ce sont les deux dernières que l'on cherche à atteindre :
  - la 1ère est impliquée par l'atomicité des valeurs,
  - la 2e n'est qu'une étape vers la 3e.

Propriétés d'une décomposition

- Soit R une relation et R_1, ..., R_n les relations issues d'une décomposition D de R :
  - D est **sans perte** si la relation R peut être recomposée par jointures des relations R_1, ..., R_n.
  - D **préserve les dépendances fonctionnelles** si les dépendances fonctionnelles de R sont impliquées par les dépendances fonctionnelles de R_1, ..., R_n.

1ère forme normale (1)

- Une relation est en **1ère forme normale** si tous ses attributs ont une valeur atomique.
- Le fait que dans le modèle relationnel les domaines soient atomiques implique que toutes les relations sont en 1ère forme normale.

1ère forme normale (2)

pas en 1FN

isbn	auteur
2-212-09283-0	Gardarin
2-7117-8645-5	Abiteboul, Hull, Vianu

en 1FN

isbn	auteur
2-212-09283-0	Gardarin
2-7117-8645-5	Abiteboul
2-7117-8645-5	Hull
2-7117-8645-5	Vianu

2e forme normale (1)

- Une relation est en **2e forme normale** si elle est en 1ère forme normale et si chaque attribut non clé dépend totalement et non partiellement de la clé primaire.

2e forme normale (2)

pas en 2FN

```
isbn bib → nb_ex
isbn → titre
```

isbn	**titre**	**bib**	**nb_ex**
2-212-09283-0	Bases de données …	Toulon	3
2-7117-8645-5	Fondements des …	Toulon	1
2-212-09283-0	Bases de données …	Aix-Marseille 3	7

en 2FN

isbn	**bib**	**nb_ex**
2-212-09283-0	Toulon	3
2-7117-8645-5	Toulon	1
2-212-09283-0	Aix-Marseille 3	7

en 2FN

isbn	**titre**
2-212-09283-0	Bases de données …
2-7117-8645-5	Fondements des …

Cette décomposition est sans perte et préserve les dépendances fonctionnelles.

Insuffisance de la 2e forme normale

en 2FN

```
isbn → titre éditeur
editeur → pays
```

isbn	**titre**	**éditeur**	**pays**
2-212-09283-0	Bases de données …	Eyrolles	France
2-7117-8645-5	Fondements des …	Vuibert	USA
2-212-0969-2	Internet/Intranet …	Eyrolles	France

Il reste des redondances : `(…, Eyrolles, France).`

3e forme normale (1)

- Une relation est en **3e forme normale** si pour chaque dépendance fonctionnelle non triviale $X \rightarrow Y$, on a :
  - soit, X est une super-clé de R,
  - soit, Y appartient à une clé candidate de R.
- La 2e partie de la règle est importante car elle dit qu'un constituant d'une clé candidate peut dépendre :
  - soit d'un constituant d'une clé candidate ,
  - soit d'un constituant non clé,

  ce qui peut être source de redondances.

3e forme normale (2)

pas en 3FN

```
isbn bib → nb_ex
isbn → titre
```

isbn	**titre**	**bib**	**nb_ex**
2-212-09283-0	Bases de données …	Toulon	3
2-7117-8645-5	Fondements des …	Toulon	1
2-212-09283-0	Bases de données …	Aix-Marseille 3	7

en 3FN

isbn	**bib**	**nb_ex**
2-212-09283-0	Toulon	3
2-7117-8645-5	Toulon	1
2-212-09283-0	Aix-Marseille 3	7

en 3FN

isbn	**titre**
2-212-09283-0	Bases de données …
2-7117-8645-5	Fondements des …

Cette décomposition est sans perte et préserve les dépendances fonctionnelles.

3e forme normale (3)

en 2FN
pas en 3FN

```
isbn → titre editeur
éditeur → pays
```

isbn	**titre**	**éditeur**	**pays**
2-212-09283-0	Bases de données …	Eyrolles	France
2-7117-8645-5	Fondements des …	Vuibert	USA
2-212-0969-2	Internet/Intranet …	Eyrolles	France

en 3FN

isbn	**titre**	**éditeur**
2-212-09283-0	Bases de données …	Eyrolles
2-7117-8645-5	Fondements des …	Vuibert
2-212-0969-2	Internet/Intranet …	Eyrolles

en 3FN

éditeur	**pays**
Eyrolles	France
Vuibert	USA

Cette décomposition est sans perte et préserve les dépendances fonctionnelles.

Insuffisance de la 3e forme normale

```
rue ville → cp
cp → ville
```

en 3FN car :
{rue, ville} est une super clé
ville appartient à une clé candidate

clés candidates :
{rue, ville}
{rue, cp}

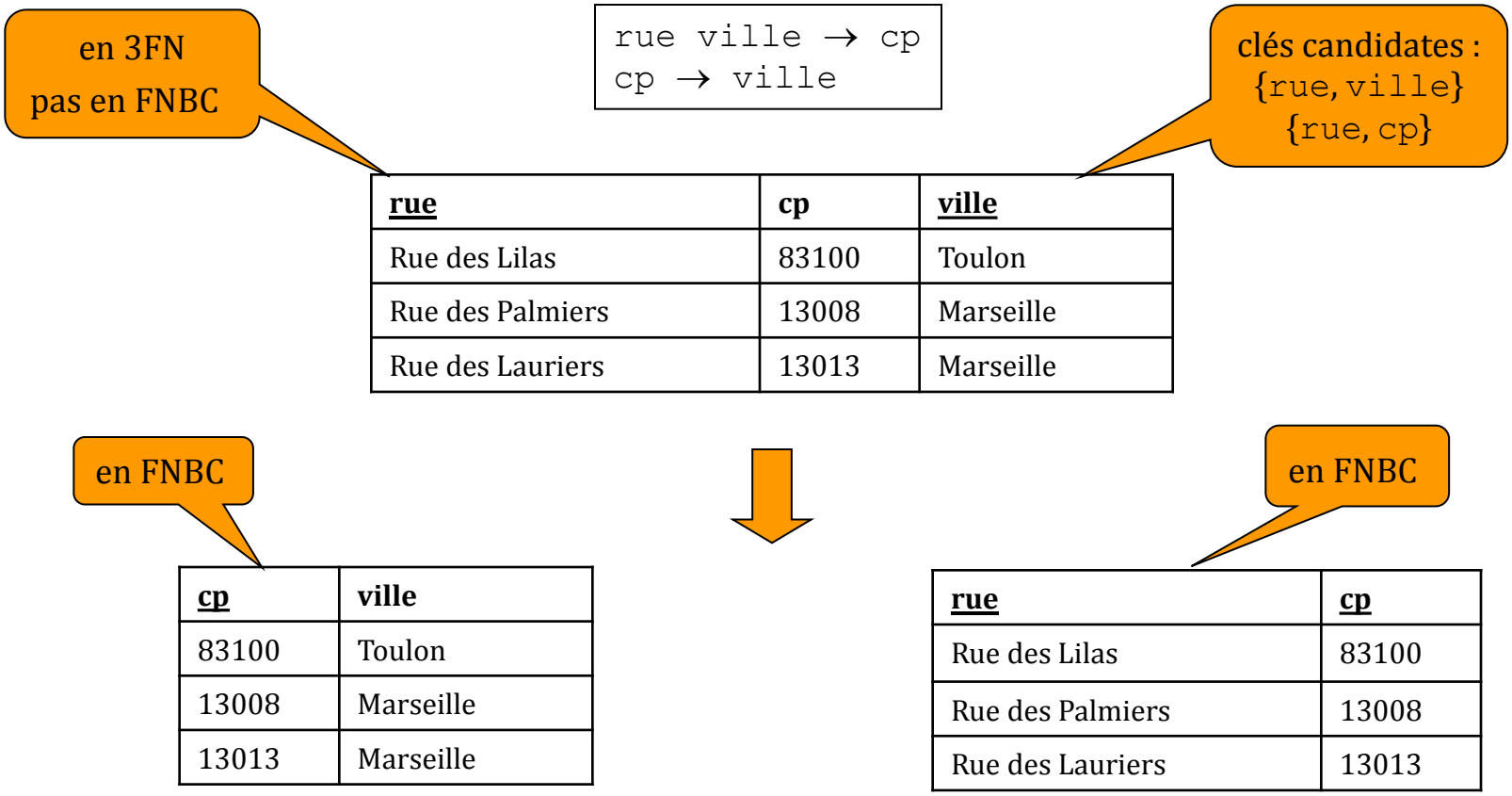
rue	cp	ville
Rue des Lilas	83100	Toulon
Rue des Pins	83100	Toulon
Rue des Palmiers	13008	Marseille
Rue des Lauriers	13013	Marseille

Il reste des redondances : (…, 83100, Toulon).

Forme normale de Boyce-Codd (1)

- Une relation R est en **forme normale de Boyce-Codd** (**FNBC**) si pour chaque dépendance fonctionnelle non triviale $X \rightarrow Y$, X est une super-clé de R.
- La forme normale de Boyce-Codd implique la 3^e forme normale.
- La forme normale de Boyce Codd est la forme idéale relativement aux dépendances fonctionnelles, mais malheureusement elle peut ne pas préserver les dépendances fonctionnelles.

Forme normale de Boyce-Codd (2)



Cette décomposition est sans perte mais **ne préserve pas les dépendances fonctionnelles**.

Décomposition en 3e forme normale et en FNBC

- Il est démontré que :
  - Toute relation a au moins une décomposition en 3e forme normale qui préserve les dépendances fonctionnelles et qui est sans perte.
  - Toute relation a au moins une décomposition en forme normale de Boyce Codd qui est sans perte mais qui peut ne pas préserver les dépendances fonctionnelles.

Dépendances multivaluées (1)

- Il y a une **dépendance multivaluée** entre un constituant X et un constituant Y d'une relation $R(X, Y, Z)$ si pour toute extension de R, à chaque valeur de X il correspond toujours le même ensemble de valeurs de Y et que cet ensemble de valeurs ne dépend pas des valeurs de Z.
- On dit que X multidétermine Y et l'on note :
  - $X \rightarrow\rightarrow Y$

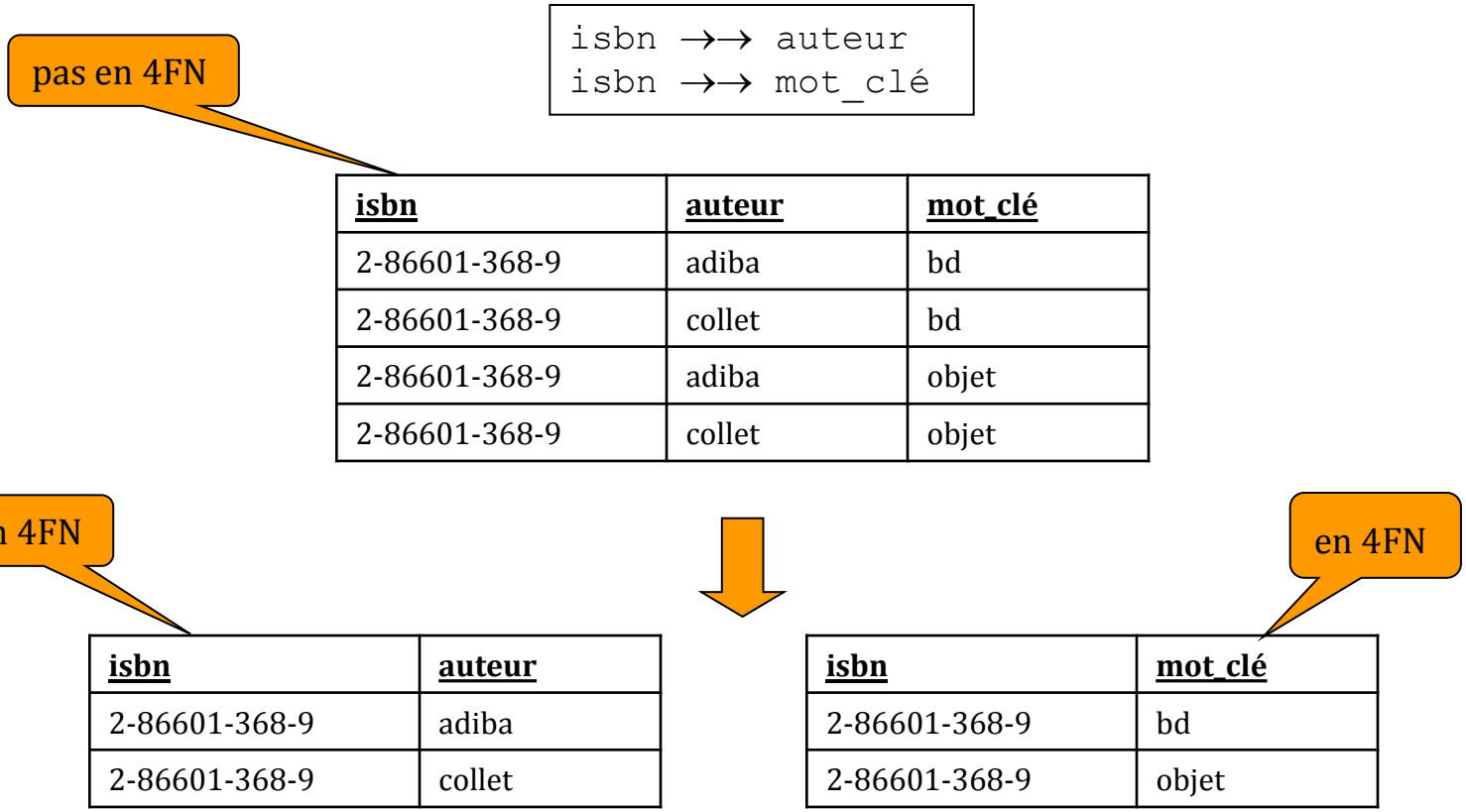
Dépendances multivaluées (2)

- Soit par exemple la relation :
  - `livre(isbn, titre, auteur)`
- Si un livre peut avoir plusieurs auteurs, la relation livre possède la dépendance multivaluée :
  - `isbn` $\twoheadrightarrow$ `auteur`
- Soit un livre d'isbn i et d'auteurs a_1, a_2 et a_3. Si le triplet (i, t, a_1) apparaît dans une extension de la relation `livre`, alors les triplets (i, t, a_2) et (i, t, a_3) doivent y apparaître aussi.

4e forme normale (1)

- Une dépendance multivaluée $X \rightarrow\rightarrow Y$ d'une relation R est dite non triviale si :
  - Y n'est pas un sous-ensemble de X,
  - $X \cup Y$ n'inclut pas tous les attributs de R.
- Une relation R est en 4e forme normale, si pour chaque dépendance multivaluée $X \rightarrow\rightarrow Y$ non triviale, X est une super-clé de R.
- La 4e forme normale implique la forme normale de Boyce-Codd puisqu'une dépendance fonctionnelle est un cas particulier de dépendance multivaluée.

4e forme normale (2)



isbn	auteur	mot_clé
2-86601-368-9	adiba	bd
2-86601-368-9	collet	bd
2-86601-368-9	adiba	objet
2-86601-368-9	collet	objet

isbn	auteur
2-86601-368-9	adiba
2-86601-368-9	collet

isbn	mot_clé
2-86601-368-9	bd
2-86601-368-9	objet

Cette décomposition est sans perte et préserve les dépendances fonctionnelles.

Décomposition en 4e forme normale (1)

- Il est démontré que toute relation a au moins une décomposition en 4e forme normale qui est sans perte mais qui peut ne pas préserver les dépendances fonctionnelles.

Décomposition en 4e forme normale (2)

- Soit U la relation à décomposer et D l'ensemble des dépendances existantes entre les attributs de U :
  - $S := \{U\}$
  - Tant qu'il existe dans S une relation R qui n'est pas en 4e forme normale :
    - on cherche dans D une dépendance $X \rightarrow\rightarrow Y$ telle que $R(X, Y, Z)$ et X n'est pas une clé de R,
    - on ajoute à Y l'ensemble Z' des attributs de Z fonctionnellement déterminés par X, produisant la dépendance $X \rightarrow\rightarrow Y\,Z'$,
    - on remplace R dans S par les deux relations $R_1(X, Y \cup Z')$ et $R_2(X, Z - Z')$.

Décomposition en 4e forme normale (3)

- S = {U(isbn, titre, auteur, mot clé, éditeur, pays, bib, nb_ex)}
  D = {isbn → titre,
  isbn → éditeur,
  éditeur → pays,
  isbn →→ auteur, isbn →→ mot_clé,
  isbn bib → nb_ex}

- S = {U1(isbn, titre, éditeur, pays),
  U2(isbn, auteur, mot clé, bib, nb_ex)}

- S = {U1(isbn, titre, éditeur, pays),
  U21(isbn, auteur),
  U22(isbn, mot clé, bib, nb_ex)}

Décomposition en 4e forme normale (4)

- S = {U1(isbn, titre, éditeur, pays),
  U21(isbn, auteur),
  U221(isbn, mot clé),
  U222(isbn, bib, nb_ex)}

- S = {U11(éditeur, pays),
  U12(isbn, titre, éditeur),
  U21(isbn, auteur),
  U221(isbn, mot clé),
  U222(isbn, bib, nb_ex)}

- Les relations de *S* sont en 4e forme normale (et donc en forme normale de Boyce-Codd) et les dépendances sont préservées.

Bilan

	3FN	FNBC	4FN
Élimination de la redondance due aux dépendances fonctionnelles	Pas toutes	Oui	Oui
Élimination de la redondance due aux dépendances multivaluées	Non	Non	Oui
Préservation des dépendances fonctionnelles	Oui	Pas toujours	Pas toujours
Préservation des dépendances multivaluées	Pas toujours	Pas toujours	Pas toujours

Organisation physique

Une organisation physique simple

- Soit la BD relationnelle :
  - `livre(titre, auteur)`
  - `personne(nom, prénom, âge)`
- La relation `livre` est stockée dans un fichier `livre.txt` dont chaque enregistrement représente un doublet de la relation `livre` et a 2 champs : les valeurs des attributs `titre` et `auteur` de ce triplet.
- La relation `personne` est stockée dans un fichier `personne.txt` dont chaque enregistrement représente un triplet de la relation `personne` et a 3 champs : les valeurs des attributs `nom`, `prénom` et `âge` de ce triplet.
- Deux métarelations décrivant les relations de la BD et leurs attributs :
  - `relation(nom, nb_att)`
  - `attribut(nom_table, nom, type, rang)`

  sont elles-même stockées dans les fichiers `relation.txt` et `attribut.txt`.

La base de données

livre	
titre	**auteur**
BD et SGBD	Dupont
XML	Durand

personne		
nom	**prénom**	**age**
Dupont	Jean	18
Durand	Pierre	20

Les fichiers

- *fichier* `table.txt`
  - personne|3
  - livre|3
- *fichier* `attribut.txt`
  - livre|titre|texte|1
  - livre|auteur|texte|2
  - personne|nom|texte|1
  - personne|texte|2
  - personne|age|entier|3
- *fichier* `livre.txt`
  - BD et SGBD|Dupont
  - XML|Durand
- *fichier* `personne.txt`
  - Dupont|Jean|18
  - Durand|Pierre|20

Evaluation d'une requête

- La requête :
  - `SELECT livre.titre`
    `FROM livre, personne`
    `WHERE livre.auteur = personne.nom AND`
    `personne.age = 30;`
- L'algorithme :
  - **pour chaque** enregistrement *l* du fichier `livre.txt`

    pour chaque enregistrement *p*
    du fichier `personne.txt`

    si *l*.3 = *p*.1 et *p*.3 = 30 **alors** afficher *l*.2;

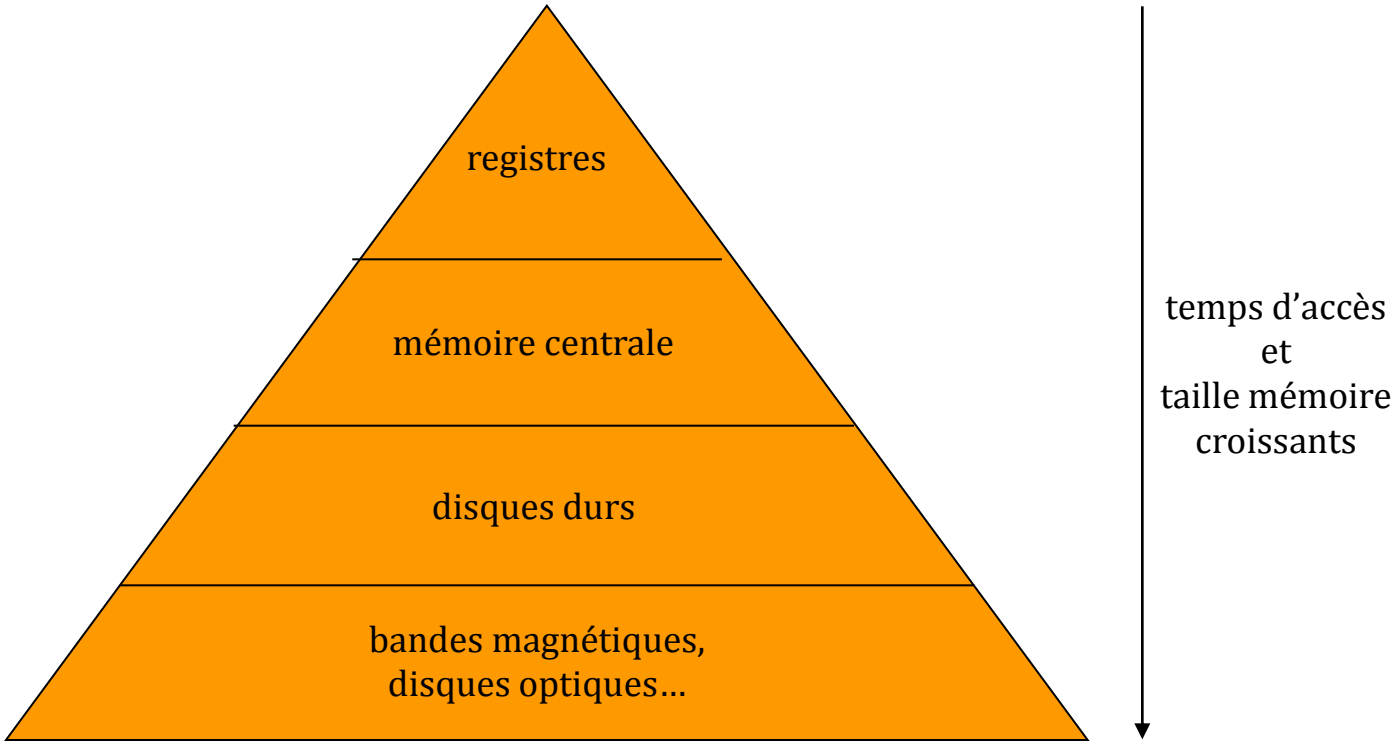
Les limites de cette organisation

- La modification d'un *n*-uplet impose de réécrire le fichier.
- Le temps d'exécution d'une requête peut être coûteux, car les relations sont parcourues séquentiellement.
- Pas de mémorisation en mémoire centrale des *n*-uplets fréquemment accédés.
- Pas de contrôle de concurrence.
- Pas de sécurité en cas de panne.

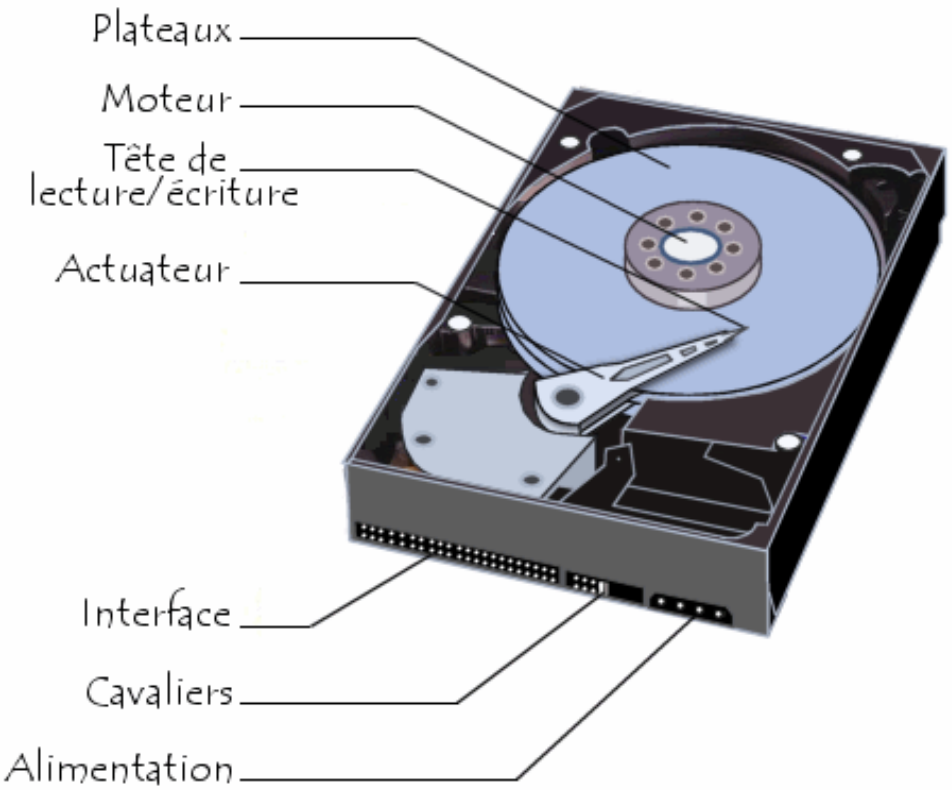
Objectifs d'une bonne organisation physique

- Une BD relationnelle est constituée d'un ensemble de relations qui ont chacune une extension qui est un ensemble de *n*-uplets.
- Physiquement ces *n*-uplets sont stockés dans un ou plusieurs fichiers qui peuvent être répartis sur un ou plusieurs sites (dans le cas de BD distribuées).
- Le format de stockage choisi doit permettre :
  - une utilisation optimale de la mémoire,
  - un accès rapide et des mises à jour peu coûteuses.
- Afin d'accéder le plus rapidement possible à un ensemble de *n*-uplets vérifiant certaines conditions, une BD relationnelle doit être munie d'index qui permettent d'associer chaque valeur d'un constituant à l'ensemble des *n*-uplets qui possèdent cette valeur pour ce constituant.

Hiérarchie de mémoire

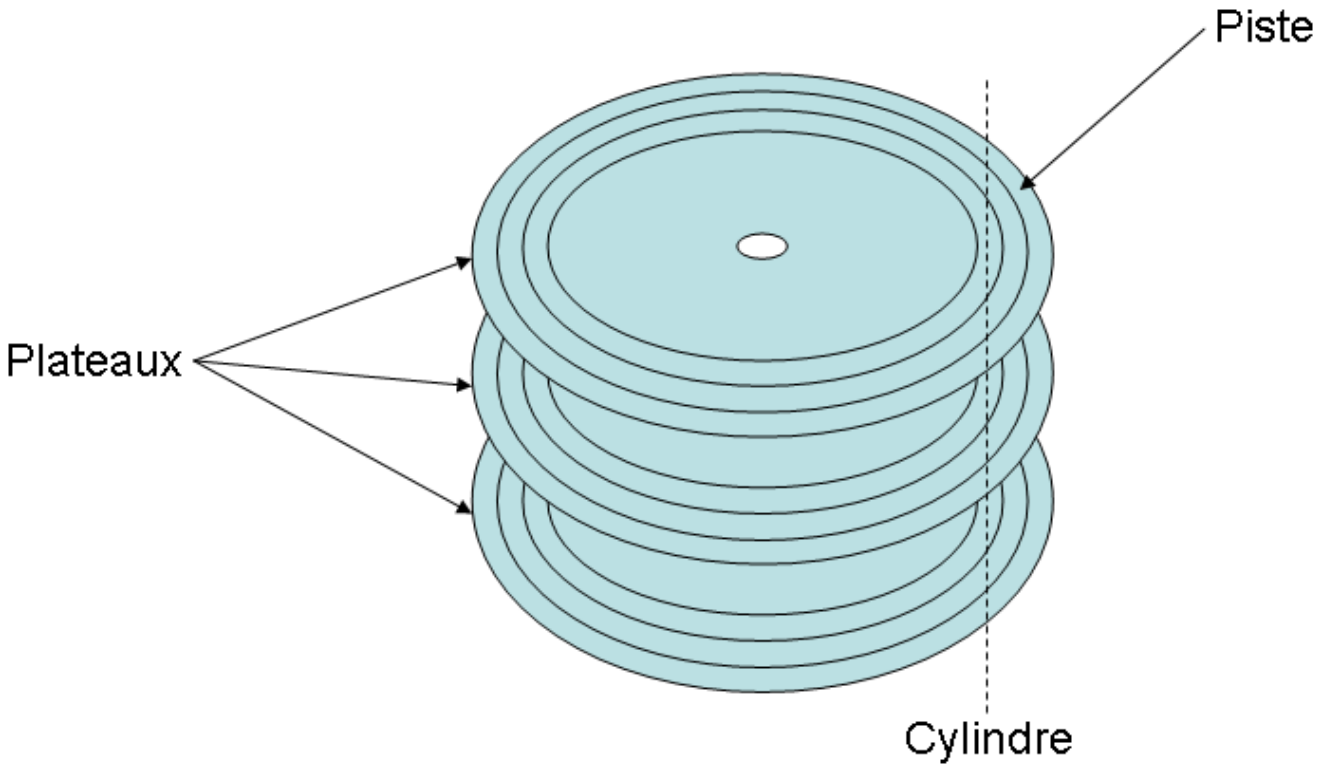


Disques durs (1)



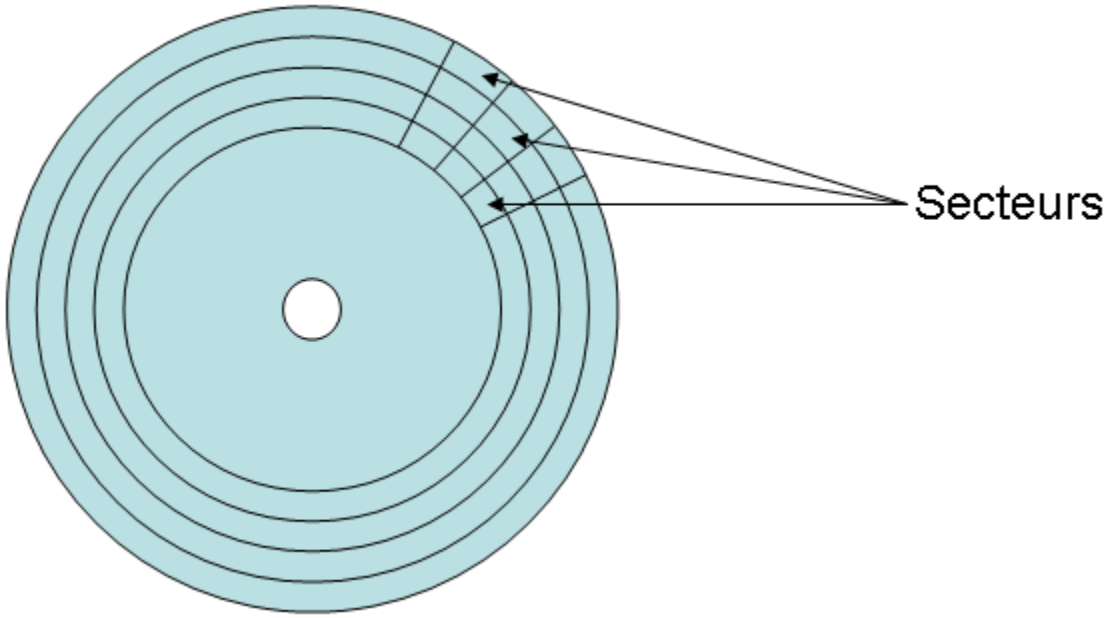
(Image extraite de http://www.commentcamarche.net/)

Disques durs (2)



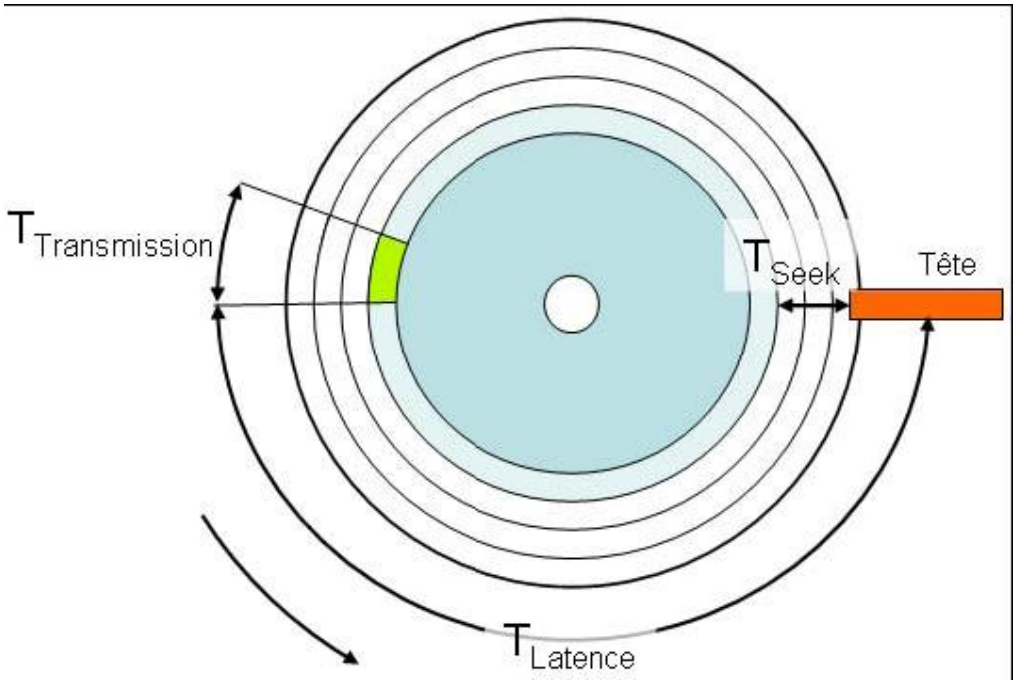
(Image extraite de Wikipédia)

Disques durs (3)



(Image extraite de Wikipédia)

Disques durs (4)



Pour un disque moderne

$T_{Latence}$ = 8,3 ms

T_{Seek} = 3,5 ms

$T_{Transmission}$ = 100 Mo/s

(Image extraite de Wikipédia)

Adresse physique d'un *n*-uplet

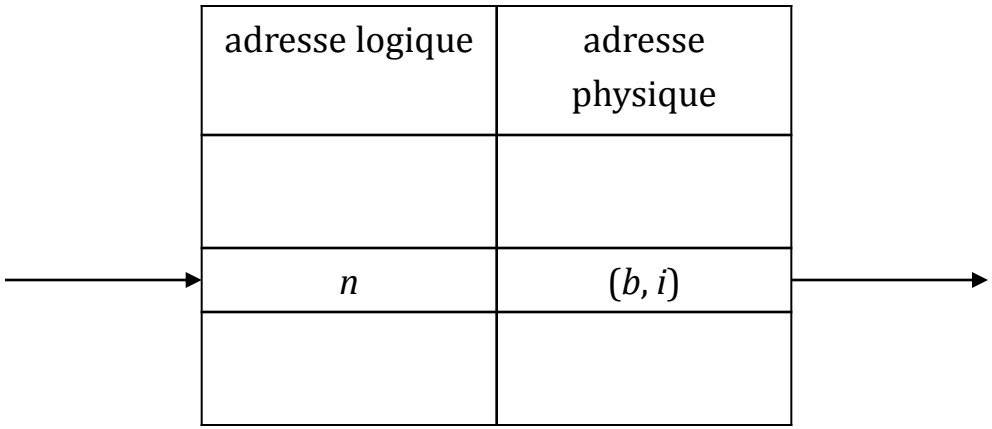
- Les *n*-uplets d'une BD relationnelle sont rangés dans des **pages.**
- Une page est stockée dans un **bloc** d'un **disque.**
- En général, un bloc est stocké sur plusieurs secteurs consécutifs du disque.
- Dans le cas général où une BD est répartie sur plusieurs **sites** et est stockée sur plusieurs disques sur un site, l'adresse physique d'un *n*-uplet est composée de :
  - l'identificateur du site,
  - l'identificateur du disque,
  - le numéro de la tête de lecture (surface),
  - le numéro de la piste,
  - le numéro du 1er secteur du bloc,
  - le début du *n*-uplet dans le bloc.

Identification d'un *n*-uplet

- Il est nécessaire d'identifier chaque *n*-uplet afin :
  - de les distinguer des autres *n*-uplets,
  - de pouvoir y accéder à partir des index.
- Deux modes d'identification peuvent être utilisés :
  - par son adresse physique,
  - par son adresse logique,
  - par combinaison des deux.

Adresse logique d'un *n*-uplet

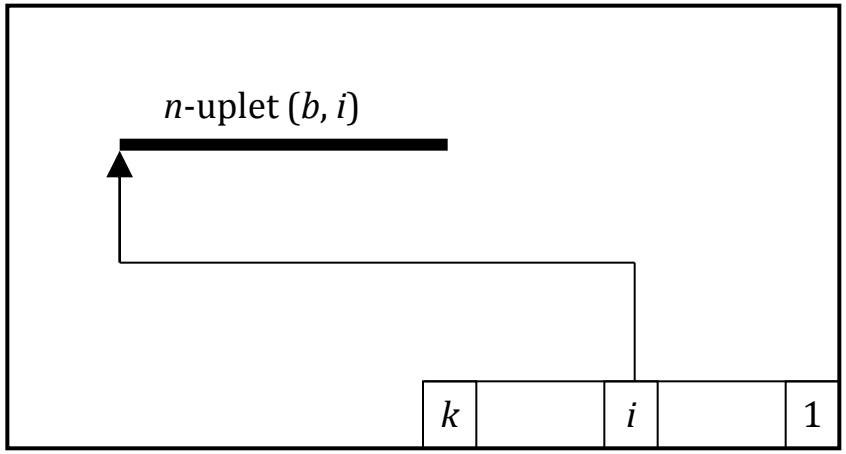
- Les *n*-uplets sont numérotés consécutivement à partir de 0 ou 1.
- Une table de correspondance permet de faire le lien entre l'adresse logique d'un *n*-uplet et son adresse physique.



Combinaison
adressage physique/adressage logique (1)

- L'adressage logique est réalisé à l'intérieur de chaque page.
- Une page est découpée en deux zones :
  - La zone des *n*-uplets : les *n*-uplets y sont implantés dans un ordre quelconque.
  - Un répertoire qui est implanté à partir de la fin de la page. La i^e case de ce répertoire indique le déplacement dans la page du *n*-uplet de rang *i*.

Combinaison
adressage physique/adressage logique (2)

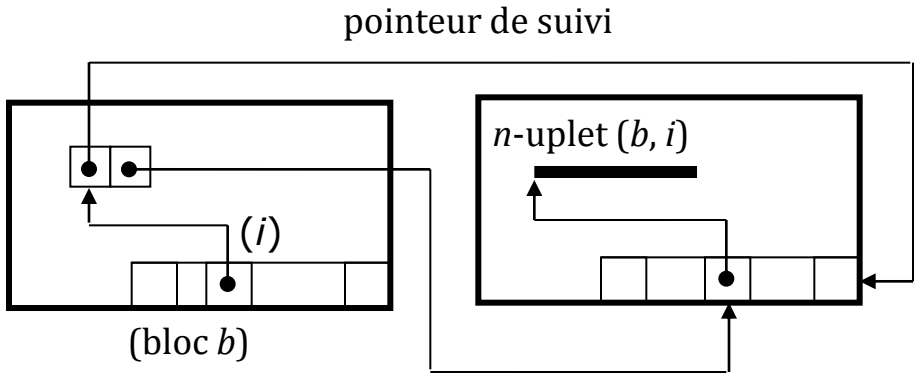


Page stockée dans un bloc b et contenant k n-uplets

Evolution de la taille des *n*-uplets

- Si la longueur d'un *n*-uplet augmente de *k* octets, deux cas sont possibles :
  - s'il reste suffisamment de place dans la page, on décale les *n*-uplets qui le suivent de k octets vers la droite, puis on change leur adresse logique dans le répertoire ;
  - sinon, on change le *n*-uplet de page et on met en place un **pointeur de suivi**.
- Si la longueur d'un *n*-uplet diminue de *k* octets, on décale les *n*-uplets qui le suivent de *k* octets vers la gauche, puis on change leur adresse logique dans le répertoire.

Pointeur de suivi



Changement de page d'un *n*-uplet (*b*, *i*) et accès à celui-ci

Représentation des valeurs d'attributs

- Elles peuvent être représentées :
  - sous leur forme externe,
  - sous la forme sous laquelle elles sont utilisées par les programmes d'application.

Représentation des *n*-uplets (1)

- Le choix d'une représentation doit prendre en compte les critères suivants :
  - possibilité de représenter des attributs de longueur variable,
  - évolutivité des schémas de relation par ajout ou suppression d'attributs,
  - rapidité d'accès aux valeurs d'attributs.

Représentation des *n*-uplets (2)

- *n*-uplets courts (taille < à celle d'une page)
  - **format fixe**
    - les valeurs d'attributs sont enregistrées dans des champs de longueur fixe,
  - **format variable**
    - les valeurs d'attributs sont stockées les unes derrière les autres :
      - soit précédées de leur longueur,
      - soit précédées de leur nom.
- *n*-uplets longs (taille > à celle d'une page)
  - les valeurs longues (images, vidéo…) sont stockées à part dans une suite consécutive de blocs,
  - le *n*-uplet est ensuite représenté comme un *n*-uplet court en remplaçant chaque valeur longue par un pointeur vers le 1er bloc la contenant.

Représentation des *n*-uplets (3)

- Soit la table :

```
CREATE TABLE personne (
    nom CHAR(15),
    prenom CHAR(10),
    age SMALLINT);
```

- Format fixe

`Dupont_____Jean_____30` (27 octets)

- Format variable

`6Dupont4Jean230` (longueur)

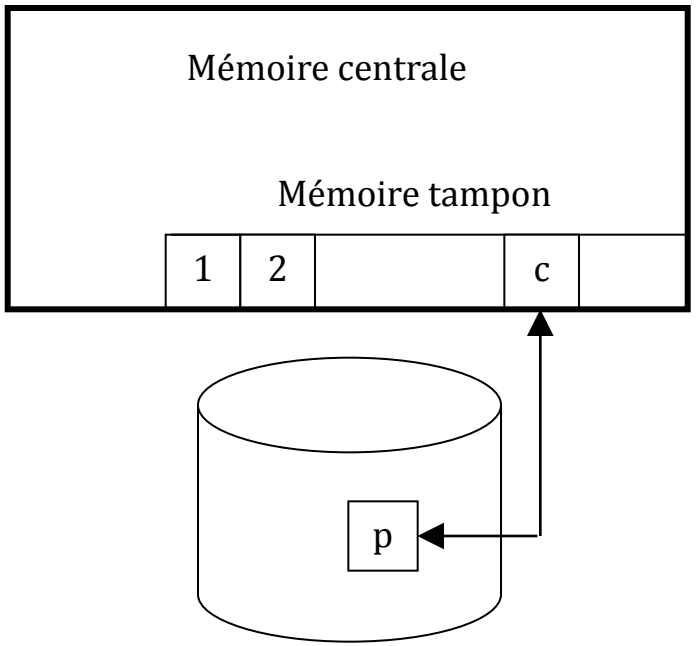
`nom#Dupont#prénom#Jean#age#18` (nom)

Echange disque/mémoire centrale

- Les échanges de données entre disque et mémoire centrale se font au travers d'une zone particulière de la mémoire appelée **mémoire tampon** (buffer).
- Les objectifs sont :
  - de rendre les pages et les *n*-uplets qu'elles contiennent accessibles en mémoire centrale,
  - de coordonner l'écriture des pages sur le disque en coopération avec le gestionnaire de transactions,
  - de minimiser le nombre d'accès disques.

Structure de la mémoire tampon

- La mémoire tampon est formée d'une suite de **cases** dont chacune peut contenir une page. En mémoire centrale une page est donc repérée par le numéro de la case du tampon dans laquelle elle est rangée.



Recherche d'une page

- L'opération de recherche d'une page :
  - a pour argument l'adresse *p* de la page cherchée,
  - retourne l'adresse *c* de la case du tampon dans laquelle cette page est rangée.
- L'algorithme est le suivant :
  - Si la page *p* est dans la case *c* du tampon, retourner *c*. On économise un accès disque.
  - Si la page *p* n'est pas dans le tampon, il faut la lire sur le disque. On teste s'il existe une case libre pour la recevoir. Si oui, soit *c* cette case.
  - Sinon, il faut libérer une case et donc renvoyer une page du tampon sur le disque. Plusieurs stratégies sont possibles.
  - Si la page à rejeter a été modifiée pendant son séjour en mémoire centrale, il faut la réécrire sur le disque. Ce travail est pris en compte par le gestionnaire de transactions.
  - Transférer la page *p* du disque dans la case *c* et retourner *c*.

Stratégies de remplacement de page

- *FIFO* (*First-In First-Out*)
  - On renvoie sur disque la page qui n'a pas été utilisée depuis le plus longtemps. Cette stratégie repose sur l'hypothèse que cette pages a moins de chances d'être réutilisée que les autres.
- *LIFO* (*Last-In First-Out*)
  - On renvoie sur disque la page qui a été utilisée le plus récemment. L'intérêt de cette stratégie est sa simplicité, car n'y a pas besoin de mémoriser les dates auxquelles les pages ont été chargées dans le tampon.
- Sous contrôle du SGBD
  - Le SGBD peut punaiser (« to pin ») des pages dans la mémoire tampon, afin qu'elles ne soient pas renvoyées sur disque, car il sait qu'elles vont être réutilisées.

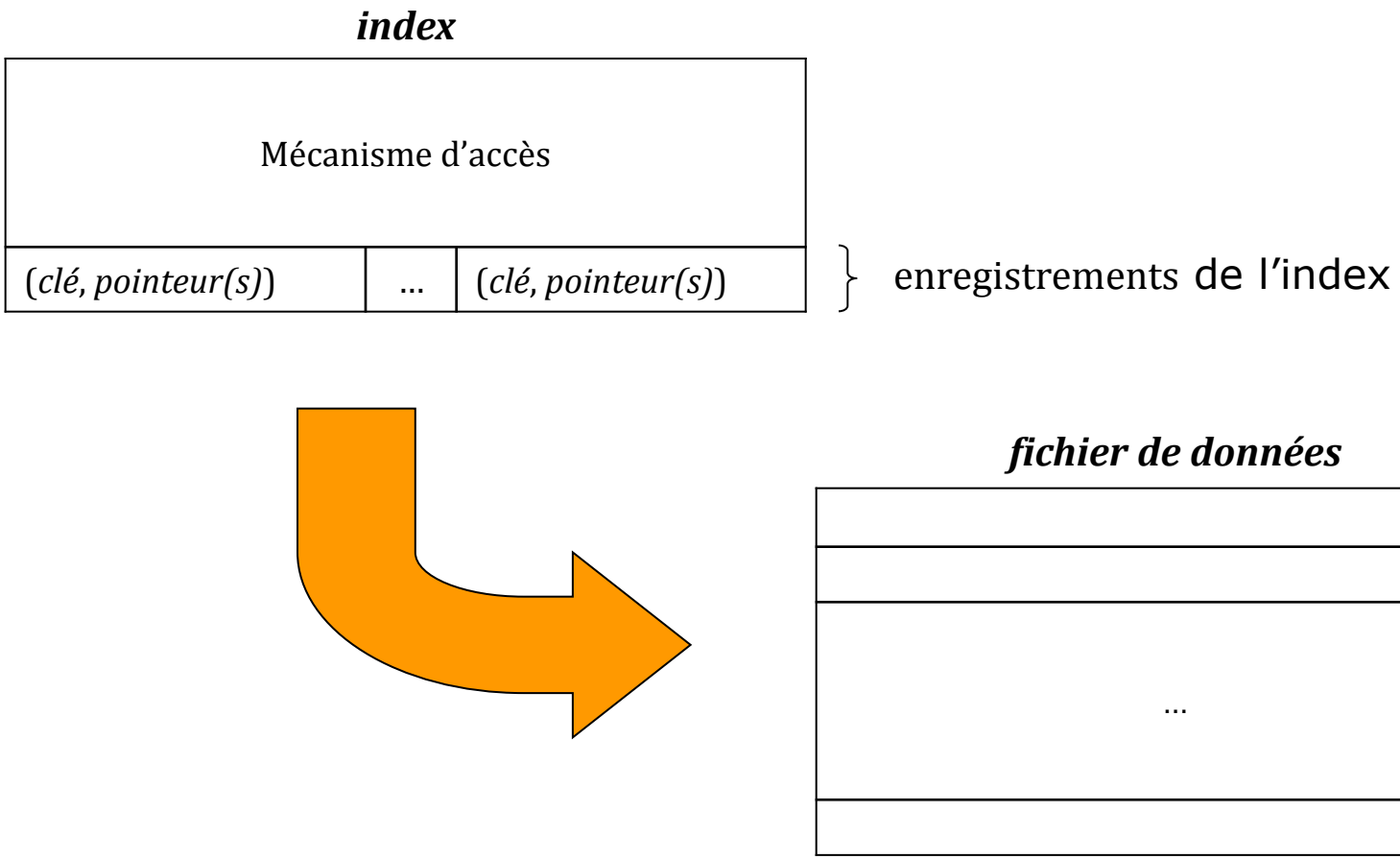
Temps d'accès à un *n*-uplet

- L'accès à un *n*-uplet consiste donc à :
  1. à rechercher sur le disque la page qui le contient,
  2. à le rechercher dans cette page.
- Le temps d'accès à un *n*-uplet est donc égal à :
  - $T_{recherche\ dans\ page} + T_{transfert\ page}$
- $T_{recherche\ dans\ page}$ est négligeable p/r à $T_{transfert\ page}$.
- Le temps d'accès à une page dépend de sa localisation (mémoire tampon ou disque).
- En conclusion, il y a intérêt :
  - à regrouper dans une même page les *n*-uplets traités consécutivement,
  - à traiter consécutivement les *n*-uplets d'une même page.

Index

- Un index est un fichier construit pour accéder de façon sélective, et donc rapide, aux enregistrements d'un fichier de données *D*, dont la valeur *c* d'un champ ou d'une liste de champs est donnée :
  - *c* est appelée une clé de recherche.
- Un index est composé de deux parties :
  - un fichier d'enregistrements à 2 champs :
    - une clé de recherche *c*,
    - un pointeur ou une liste de pointeurs vers des enregistrements de *D*,
  - un mécanisme d'accès à un enregistrement de l'index à partir de la clé de recherche.

Structure d'un index



Index primaire/index secondaire

- Un index est dit :
  - **primaire** si la clé de recherche est une clé d'un enregistrement du fichier de données :
    - en ce cas, un enregistrement de l'index pointe vers un seul enregistrement du fichier de données.
  - **secondaire** sinon :
    - en ce cas, un enregistrement de l'index peut pointer vers plusieurs enregistrements du fichier de données.

Index dense/index creux

- Un index est dit :
  - **dense** si chaque clé de recherche dans le fichier de données apparaît dans l'index.
  - **creux** si seulement certaines clés du fichier de données apparaissent dans l'index (en général, une clé par bloc du fichier de données).
- Un index dense est dit :
  - **groupant**, si ses enregistrements et ceux du fichier de données sont triés selon la clé de recherche.
  - **non groupant**, sinon.
- Un index secondaire est toujours dense.

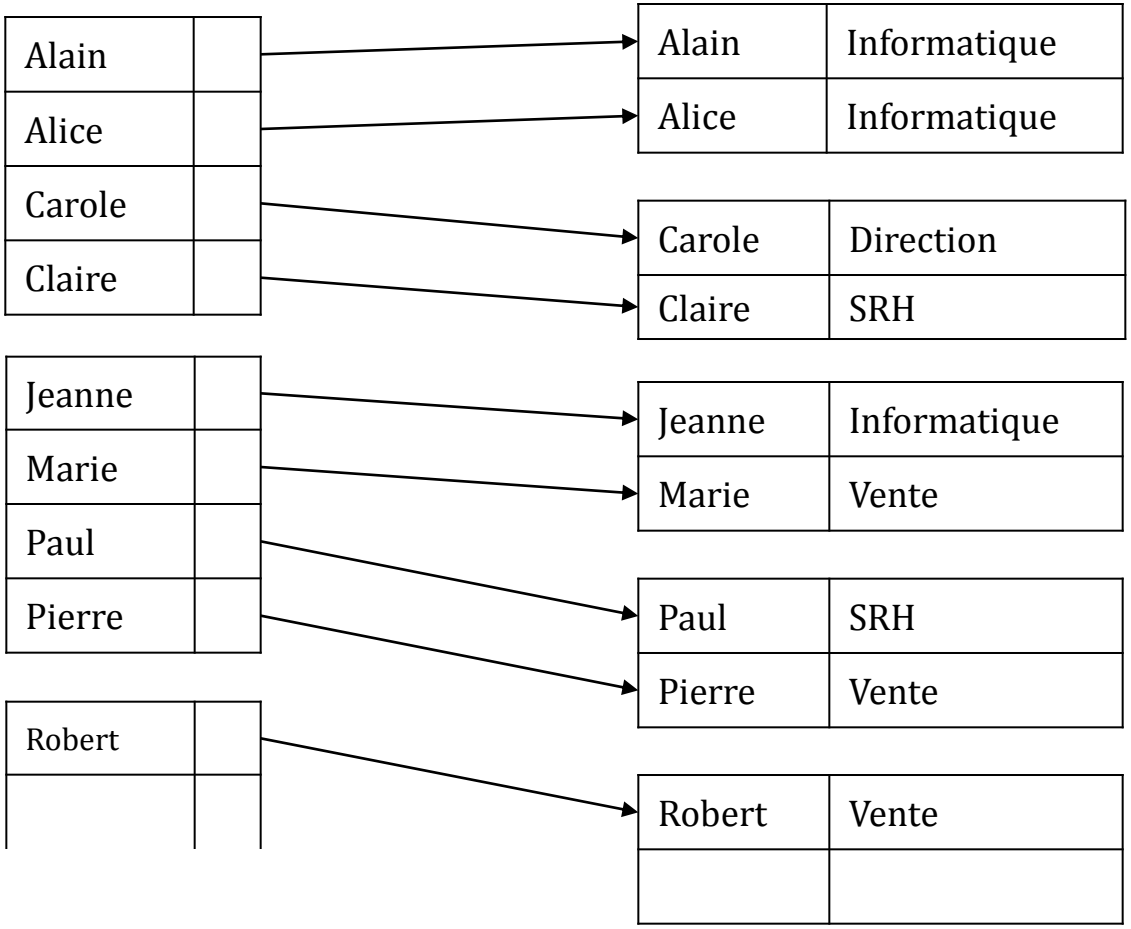
Index dans les BD relationnelles

- Dans une BD relationnelle, les index sont construits sur une relation, par la commande SQL :
  - `CREATE INDEX` *nom* `ON` *relation* (*liste d'attributs*)
- Pour une relation *R* on distingue :
  - l'**index primaire** construit sur la clé primaire de *R* :
    - il donne accès au *n*-uplet dont la clé primaire est donnée, ou à son identificateur.
  - les **index secondaires** construits sur un attribut ou une liste d'attributs de *R* :
    - ils donnent accès aux identificateurs des *n*-uplets dont la valeur pour ce constituant est donnée.

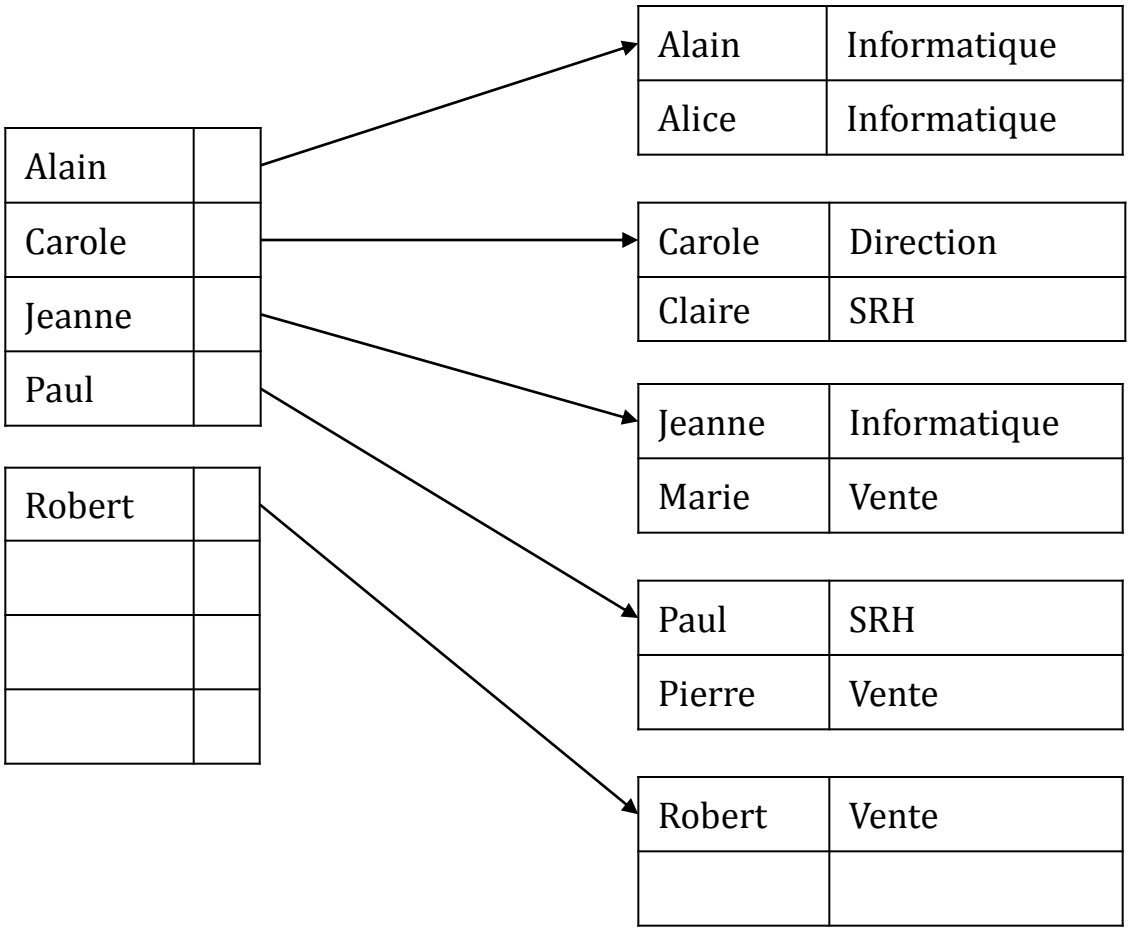
Exemple : la table

employé	
<u>nom</u>	**département**
Alain	Informatique
Pierre	Vente
Marie	Vente
Jeanne	Informatique
Carole	Direction
Paul	SRH
Alice	Informatique
Claire	SRH
Robert	Vente

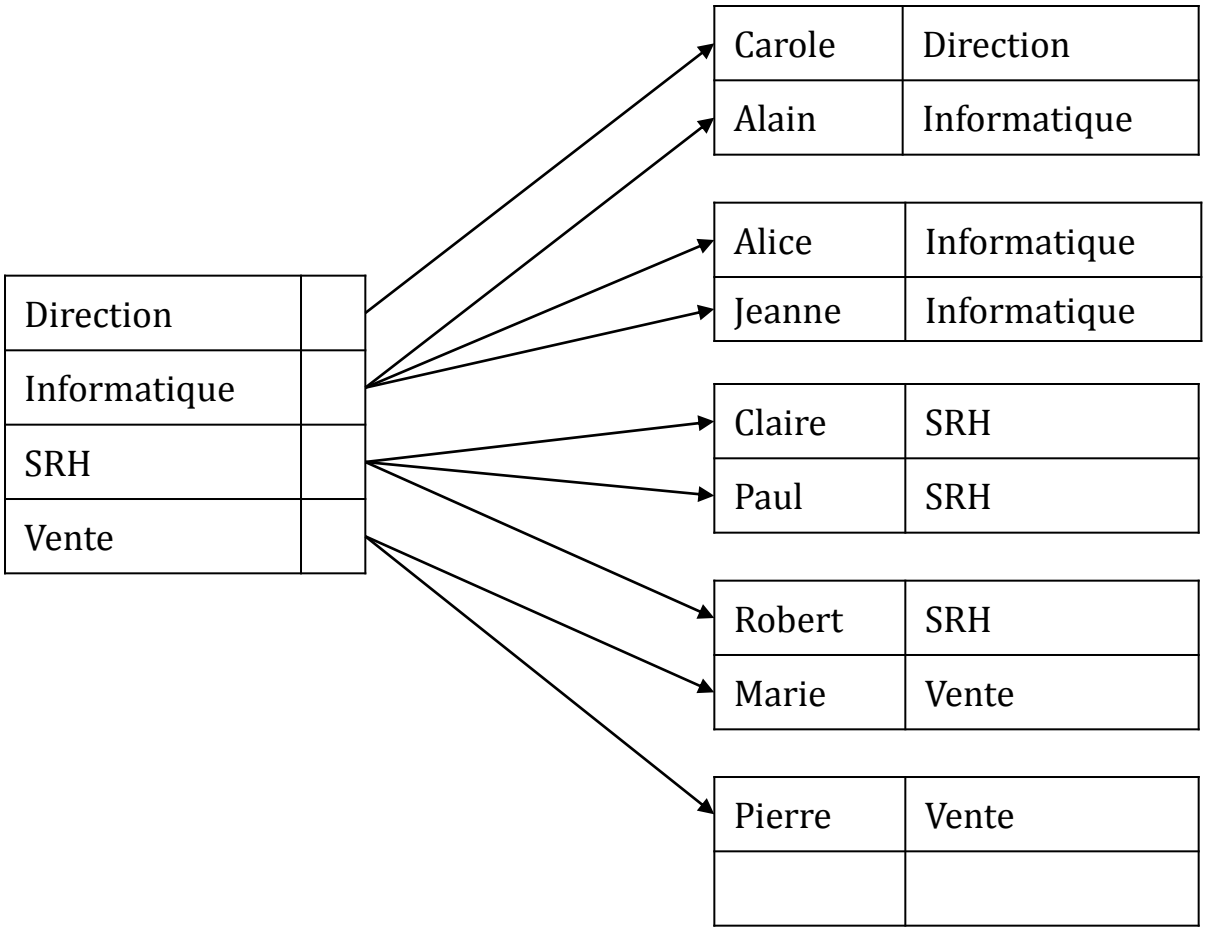
Exemple : index primaire dense



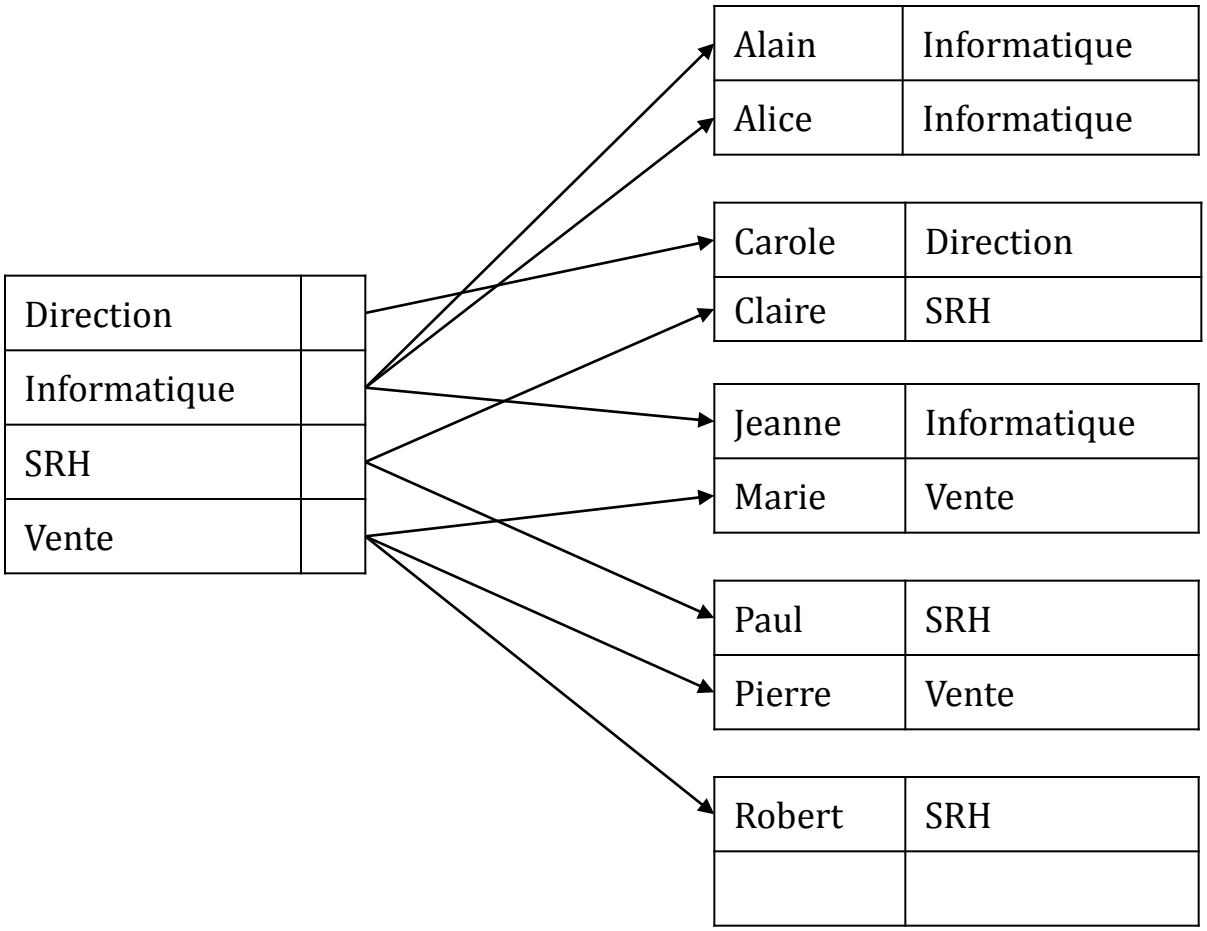
Exemple : index primaire creux



Exemple : index secondaire groupant



Exemple : index secondaire non groupant



Mécanisme d'accès aux enregistrements d'un index

- Organisation arborescente
  - séquentiel-indexé,
  - arbres B+
- Accès par hachage
  - statique
  - dynamique

Exemple : la table sur laquelle sera construit l'index

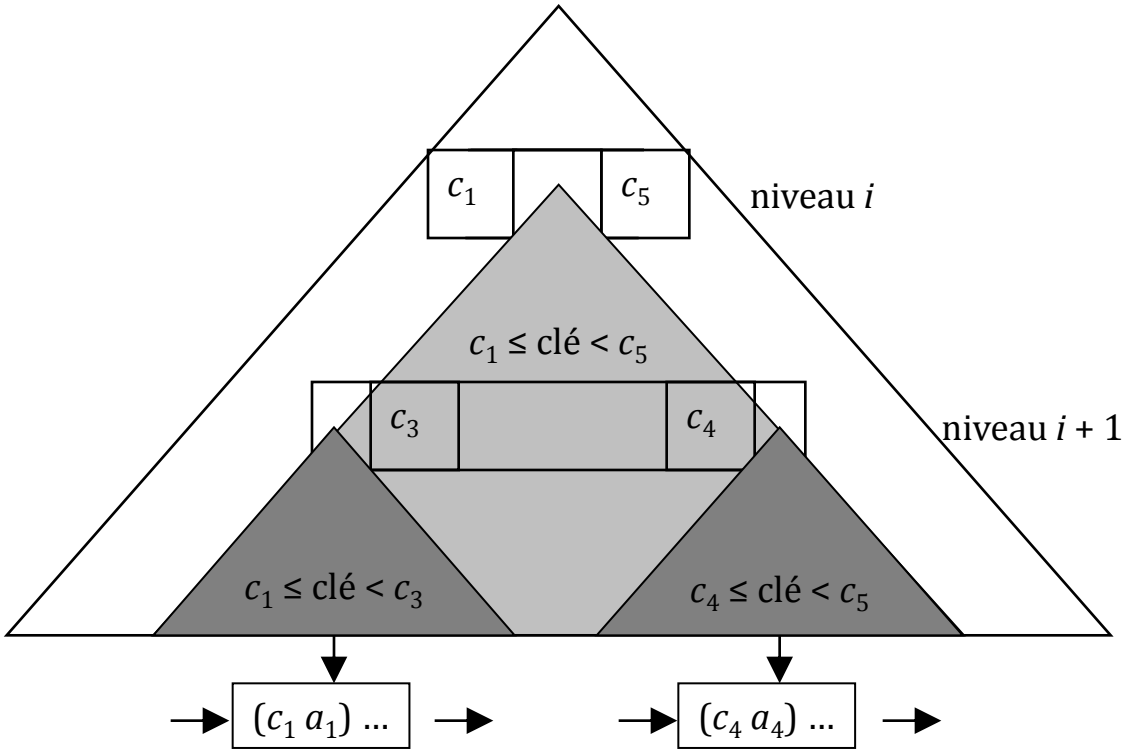
Chaque mécanisme d'accès sera illustré par la construction de l'index primaire, que nous appellerons `dico`, de la relation suivante :

dictionnaire	
mot	**définition**
mélodie	Suite de sons formant un air...
école	Etablissement où se donne un enseignement collectif…
nez	Partie saillante du visage...
bateau	Nom des embarcations, des navires...
kayak	Embarcation étanche et légère...
zébu	Bœuf à longues cornes et à bosse sur le garrot…
dessin	Représentation sur une surface de la forme d'un objet...
corde	Assemblage de fils tressés ou tordus ensemble…
terre	Planète habitée par l'homme…

Arbre B+

- Les arbres B ont été introduits par Bayer et McCreight en 1972 et ont fait l'objet de nombreux développements par la suite.
- Nous décrivons une variante des arbres B : les arbres B+, qui sont très largement utilisés pour construire des index de BD.

Structure d'un arbre B+



Organisation d'un arbre B^+ (1)

- Un arbre B^+ d'ordre *m* (entier impair ≥ 3) est un arbre équilibré dont chaque nœud est enregistré dans une page stockée sur disque.

Organisation d'un arbre B^+ (2)

- Une feuille contient une séquence d'enregistrements :
  - $(c_1\ a_1) \ldots (c_k\ a_k)\ p$

  où :
  - c_i est une clé et a_i est l'information associée à cette clé,
  - p est un pointeur vers la feuille suivante,

  triée par ordre croissant de clé.
- Une feuille est au moins à moitié remplie :
  - $(m + 1) / 2 \leq k \leq m$

  sauf si elle est l'unique nœud de l'index.
- Les feuilles sont chaînées entre elles dans l'ordre de leur première clé à l'aide du pointeur p.

Organisation d'un arbre B^+ (3)

- Un nœud non terminal contient une séquence d'enregistrements :
  - $(p_1) \ldots (c_k\, p_k)$

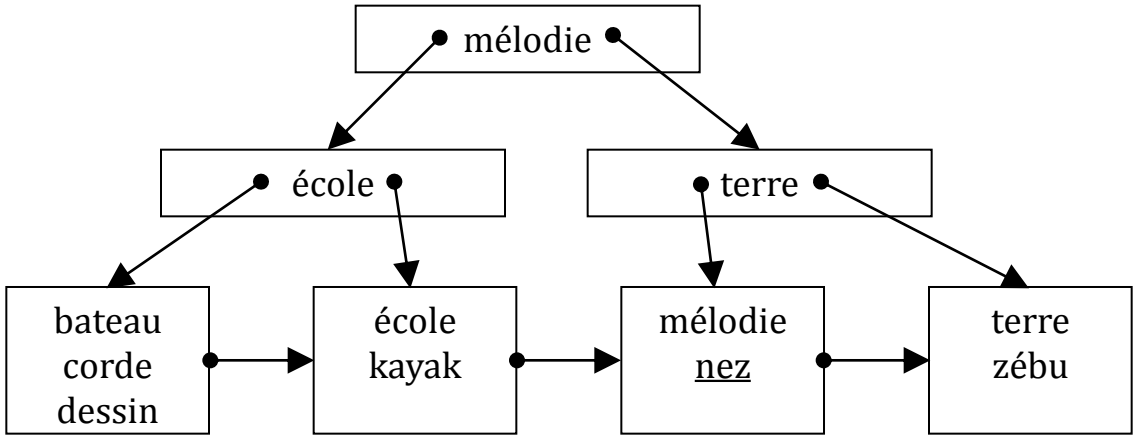
  où :
  - c_i est une clé,
  - p_i $(1 \leq i \leq m - 1)$ est un pointeur vers la racine du sous-arbre dont les feuilles contiennent les enregistrements dont la clé est égale ou supérieure à ci et inférieure à $c_i + 1$.
  - p_m est un pointeur vers la racine du sous-arbre dont les feuilles contiennent les enregistrements dont la clé est égale ou supérieure à cm.

  triée par ordre croissant de clé.
- Un nœud est au moins à moitié rempli
  - $(m + 1) / 2 \leq k \leq m$

  sauf s'il est la racine, auquel cas son contenu peut se réduire à :
  - $(p_1) \ldots (c_2\, p_2)$.

L'index `dico` en arbre B+

L'arbre B+ est d'ordre $m = 3$.
Seules les clés des enregistrements sont représentées.



Recherche d'un enregistrement

- (Il s'agit de rechercher l'enregistrement dont la clé c est donnée ou, s'il n'existe pas, le nœud qui devrait le contenir.)

  début

  Le nœud courant est la racine de l'arbre B+.

  tant que le nœud courant est un nœud non terminal de contenu (p_1) (c_2, p_2)… (c_n, p_n) **répéter**

  si $c < c_2$ **alors**

  Accéder au nœud d'adresse p_1 qui devient le noeud courant.

  sinon

  Rechercher séquentiellement le dernier c_i inférieur ou égal à c et accéder au nœud d'adresse p_i qui devient le nœud courant.

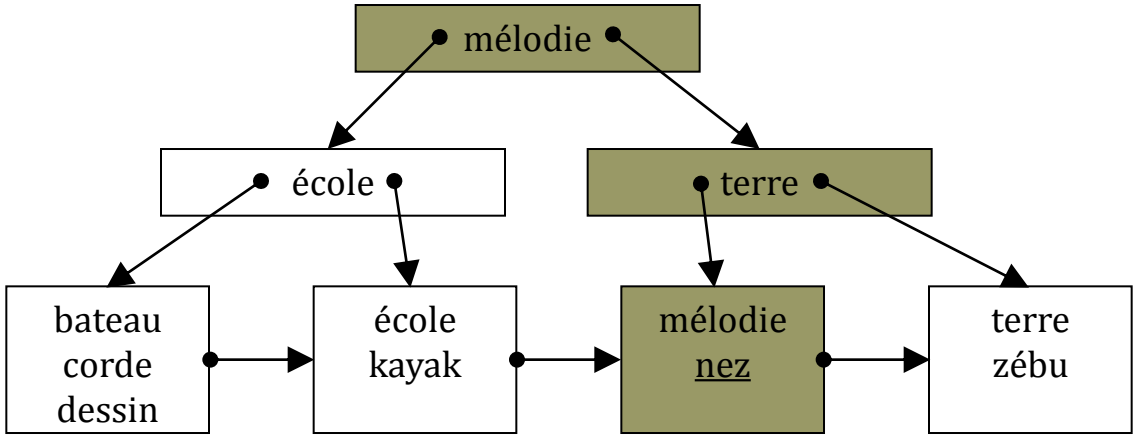
  fin

  fin

  Rechercher l'enregistrement de clé c dans le nœud courant (une feuille).

  fin

Recherche de `nez` dans l'index `dico`



Insertion

- (Il s'agit d'insérer l'enregistrement de clé *c* et d'information associée a.)
  début
    Rechercher l'enregistrement de clé *c*.
    si il existe **alors**
      L'insertion est terminée.
    sinon
      Le nœud courant est celui sur lequel s'est arrêtée la recherche
      et l'enregistrement à insérer est (*c*, *a*).
      répéter
        Soit *n* le nœud courant, *p* son adresse et *s* le contenu de *n*.
        Insérer l'enregistrement à insérer dans *s* en respectant l'ordre des clés.
        si *longueur*(*s*) ≤ *m* **alors**
          Enregistrer *s* dans *n*.
          L'insertion est terminée.
        sinon
          fission du nœud courant
        fsi
      jusqu'à ce que l'insertion soit terminée
    fsi
  fin
  (La longueur d'une séquence est le nombre d'enregistrements qui la composent.)

Fission d'un nœud

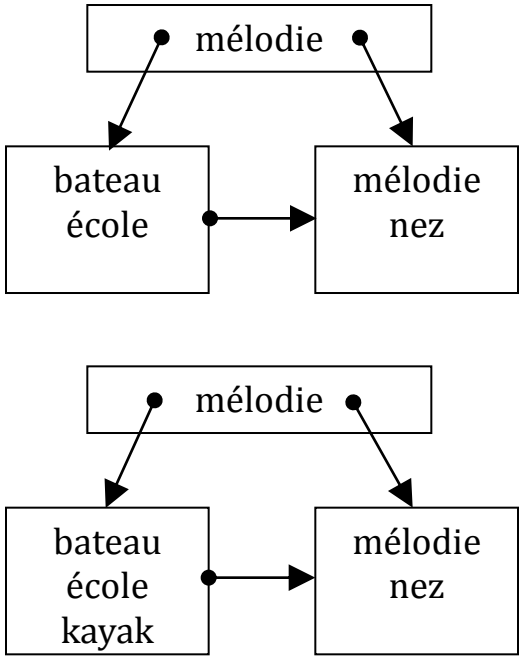
- (Il s'agit de répartir sur deux nœuds le nouveau contenu s du nœud n d'adresse p)
  début
    Créer un nouveau nœud *n*' d'adresse *p*'.
    Découper *s* en deux séquences s_1 et s_2 de longueur égale :
    c'est possible, car la longueur de s est $m + 1$ qui est un nombre pair.
    Soit c_{21} la première clé de s_2.
    Enregistrer s_1 dans *n*.
    Enregistrer s_2 dans *n*' après avoir supprimé sa première clé,
    si *n*' est un nœud non terminal.
    si *n* est une feuille **alors**
      Chaîner *n*' à *n*.
    fin
    si *n* est la racine **alors**
      Créer un nouveau nœud de contenu (*p*) (c_{21} *p*').
      L'insertion est terminée (la hauteur de l'index a augmenté de 1).
    sinon
      Le père de n devient le nœud courant
      et (c_{21} *p*') devient l'enregistrement à insérer.
    fsi
  fin

Construction de l'index `dico` (1)

Insertions successives des enregistrements : mélodie, école, nez.

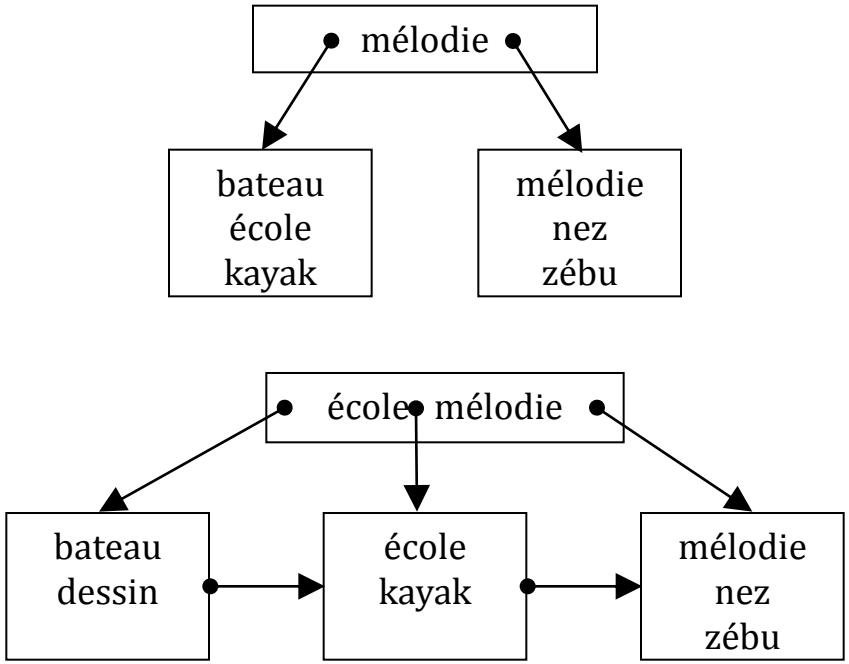
Construction de l'index `dico` (2)

Insertions successives des enregistrements : bateau, kayak.



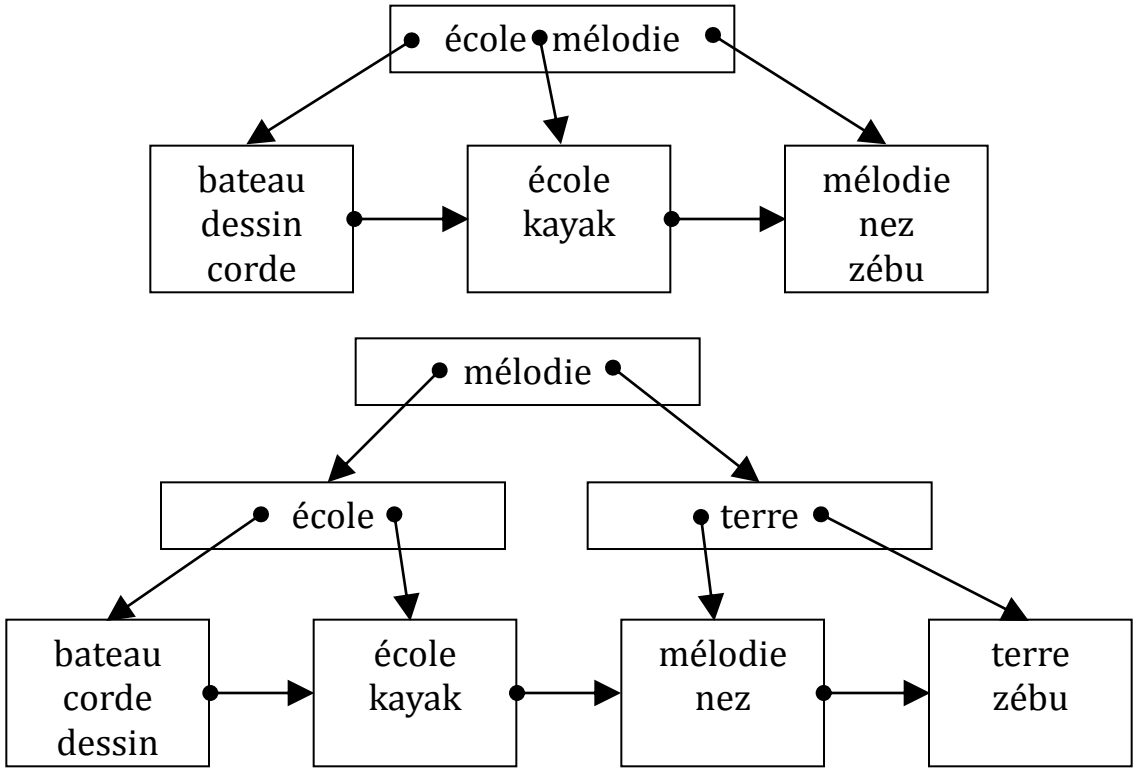
Construction de l'index `dico` (3)

Insertions successives des enregistrements : zébu, dessin



Construction de l'index `dico` (4)

Insertions successives des enregistrements : corde, terre.



Suppression d'un enregistrement

- (Il s'agit de supprimer l'enregistrement dont la clé c est donnée.)
  début
    Rechercher l'enregistrement de clé *c*.
    si il n'existe pas **alors**
      La suppression est terminée.
    sinon
      Le nœud courant est celui qui contient l'enregistrement de clé *c*.
      répéter
        Soit *n* le nœud courant, *p* son adresse, *s* le contenu de *n*.
        Supprimer l'enregistrement à supprimer de *s*.
        si *longueur*(*s*) ≥ (*m* + 1) / 2 ou si *n* est la racine **alors**
          Enregistrer *s* dans *p*.
          La suppression est terminée.
        sinon
          Fusion du nœud courant
        fsi
      jusqu'à ce que la suppression soit terminée.
    fsi
  fin

Fusion d'un nœud (1)

- (Il s'agit de fusionner le nouveau contenu s du nœud courant avec le contenu de son frère gauche ou de son frère droit.)

  début

  si la fusion se fait avec le frère gauche **alors**

  Soit n_2 le nœud courant et n_1 son frère gauche.

  Ré-associer sa clé au premier enregistrement de s,
  si n_2 est un nœud non terminal.

  Soit s' la concaténation du contenu de n_1 avec s.

  sinon (la fusion se fait avec le frère droit)

  Soit n_1 le nœud courant et n_2 son frère droit.

  Ré-associer sa clé au premier pointeur du contenu de n_2,

  si n_2 est un nœud non terminal.

  Soit s' la concaténation de s avec le contenu de n_2.

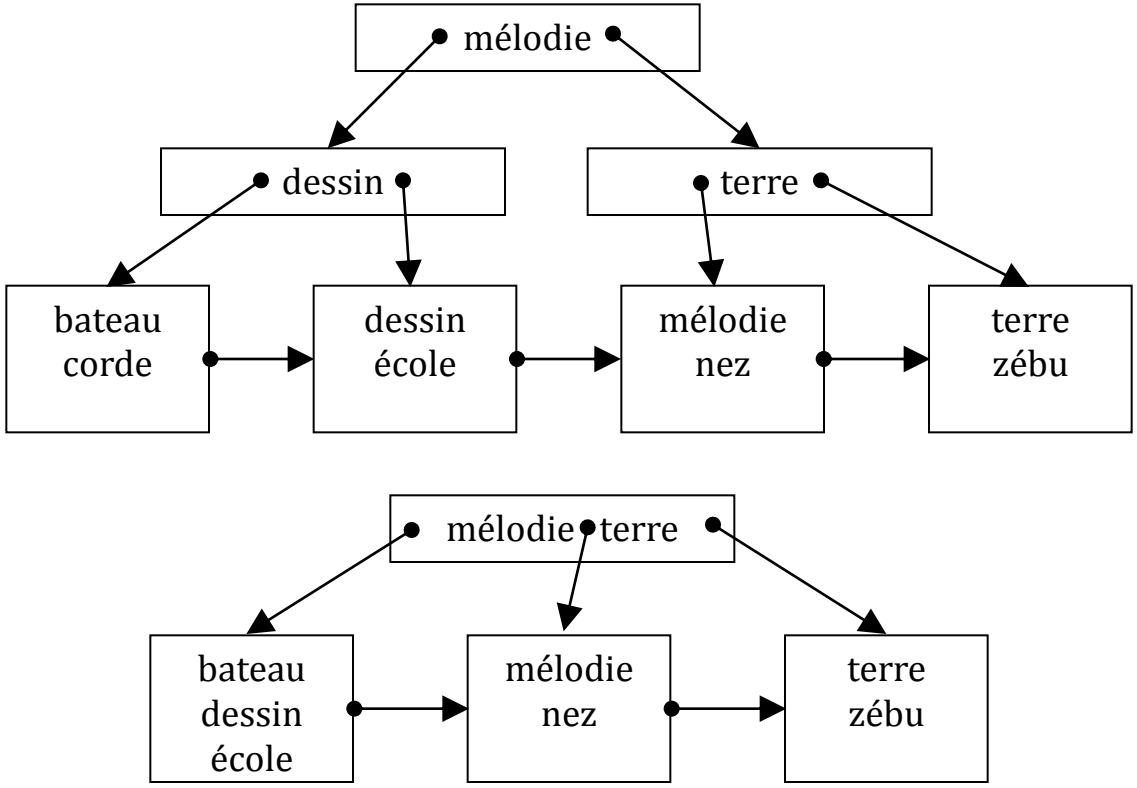
  ...

Fusion d'un nœud (2)

...
si *longueur*(*s*') ≤ *m* **alors**
    Enregistrer *s*' dans n_1.
    Supprimer n_2.
    si n_1 est fils unique **alors**
        Supprimer le père de n_1.
        La suppression est terminée (la hauteur de l'index a diminué de 1).
    fsi
    Le père de n_2 devient la page courante et l'enregistrement de ce nœud qui pointe vers n_2 devient l'enregistrement à supprimer.
sinon
    Découper *s*' en deux séquences s'_1 et s'_2 de longueurs égales à un item prêt.
    Enlever sa clé au premier item de s'_2, si n_2 est un nœud non terminal.
    Enregistrer s'_1 dans n_1 et s'_2 dans n_2.
    Remplacer dans le nœud père de n_2, la clé de l'enregistrement qui pointe vers n_2, par la première clé de s'_2.
    La suppression est terminée.
fsi
fin

Suppressions dans l'index `dico`

Suppressions successives des enregistrements : kayak, corde



Performances (1)

- Le nombre de lectures de nœuds pour accéder à un enregistrement est égal à la longueur d'une branche,
  c.-à-d. à la hauteur de l'arbre.
- La hauteur maximum (*hmax*) de l'arbre est obtenue quand la racine est réduite à deux enregistrements et les autres nœuds ne sont remplis qu'à moitié, c.-à-d. ne contiennent que (m + 1) / 2 enregistrements.
- Calculons *hmax* en posant $n = (m + 1) / 2$:
  - au 1er niveau, l'arbre possède 2 enregistrements,
  - au 2e niveau, il en possède $2n$,
  - …
  - au i^e niveau, il en possède $2n^{i-1}$.
- Soit N le nombre d'enregistrements de l'index. On a :
  - $2n^{hmax-1} = N$ et donc $hmax = log_n(N / 2) + 1$

Performances (2)

- Par exemple, si l'on suppose :
  - que l'on peut ranger 99 enregistrements par noeuds,
  - qu'il y a 10^6 enregistrements dans l'index,

  alors $m = 99$, $n = 50$.
- On a : $\log_{50}(106/2 + 1) \approx 3{,}85$.
- Il faut donc 4 lectures de bloc disque dans le pire cas pour retrouver un enregistrement à partir de sa clé.

Hachage

- Le **hachage** repose sur la construction d'une fonction dite de hachage qui appliquée à la clé d'un enregistrement fournit l'adresse de cet enregistrement.
- Le hachage est dit **statique** ou **dynamique** selon la fonction de hachage est fixée ou évolue durant la vie de l'index.
- Un index à accès par hachage peut être organisé avec ou sans **répertoire**.

Hachage statique

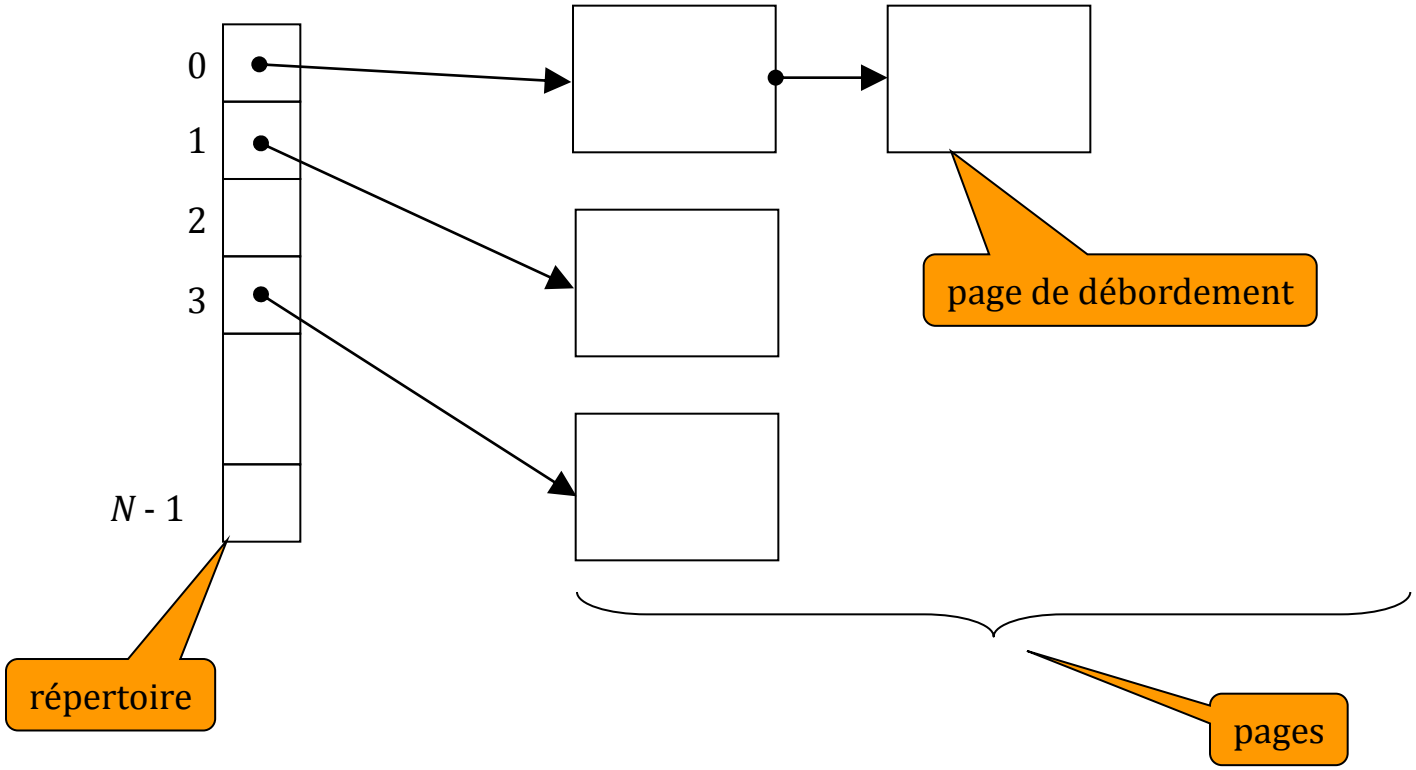
- Nous présentons en détail l'organisation avec répertoire puis, sur un exemple, l'organisation sans répertoire.

Organisation (1)

- Un index à accès par hachage statique avec répertoire est un quadruplet (N, h, P, R) où :
  - N est un nombre entier positif,
  - h est une **fonction de hachage** qui appliquée à une clé produit un nombre entier compris entre 0 et N - 1, appelé **code haché** (« hash-code »).
  - P est un ensemble de pages stockées sur disque

    Chaque page contient une liste d'enregistrements (c a) où :
    - c est la clé,
    - a est l'information associée.
  - R est un répertoire de N cases.
    Chaque case i ($0 \leq i \leq N - 1$) est le début d'une chaîne (éventuellement vide) de pages dont tous les enregistrements sont tels que $H(c) = i$.
  - Les pages de cette chaîne à partir de la 2ème position sont appelées pages de **débordement** (« overflow »).

Organisation (2)



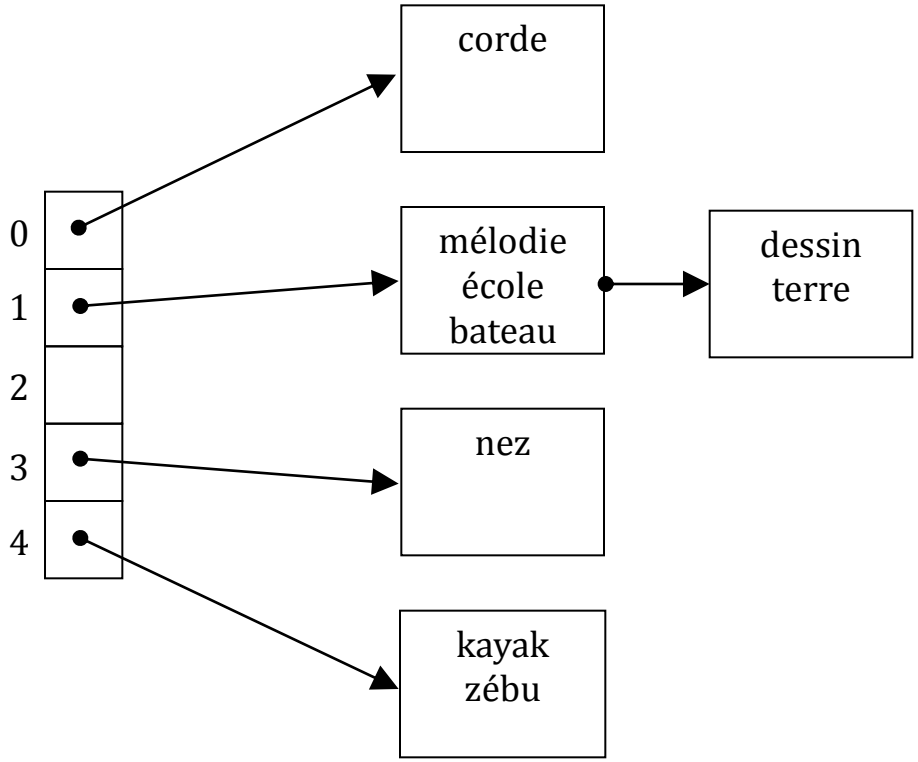
Fonction de hachage

- La fonction de hachage s'applique à une clé, que nous supposerons être une chaîne de caractères, et à un nombre entier N. Elle retourne un nombre entier compris entre 0 et N - 1.
- Il n'est en général pas possible général de construire une fonction de hachage injective, c'est à dire, telle que :
  - $c_1 \neq c_2 \Rightarrow h(c_1) \neq h(c_2)$
- On construit donc une fonction qui minimise le nombre de collisions et les répartit uniformément.
- Les deux techniques les plus utilisées sont la division et le pliage.
- Dans les deux cas, on construit à partir des codes des caractères de la clé c, un nombre k grand devant N. La valeur de $h(c)$ est alors calculée comme suit :
  - **Division**. h(c) est le reste de la division de k par N. Il est montré que l'on a intérêt à choisir N premier.
  - **Pliage**. On découpe la représentation binaire de k en tranches de b bits. $h(c)$ est égal au « ou exclusif » des nombres binaires ainsi obtenus.

L'index `dico` par hachage avec répertoire

$N = 5$: répertoire de 5 cases,
3 enregistrements au maximum,
$h(c)$ = (somme des codes ASCII des caractères de c) modulo 5.

c	***h*(c)**
mélodie	1
école	1
nez	3
bateau	1
kayak	4
zébu	4
dessin	1
corde	0
terre	1



Recherche d'un enregistrement

- (Il s'agit de rechercher l'enregistrement dont la clé c est donnée.)

  début

    Parcourir les pages liées à la case $h(c)$

    jusqu'à trouver une page qui contienne un

    enregistrement de clé c :
    c'est l'enregistrement recherché.

    si la fin de la chaîne est atteinte **alors**

      L'enregistrement recherché n'existe pas.

    fsi

  fin

Insertion d'un enregistrement

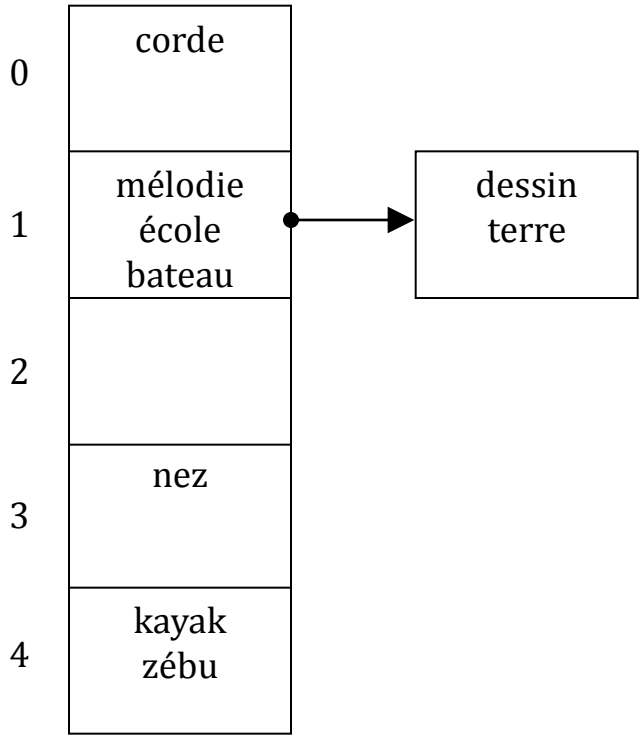
- (Il s'agit d'insérer un nouvel enregistrement de clé c et d'information associée a.)
  début
    Rechercher l'enregistrement de clé c.
    si il existe **alors**
      L'insertion est terminée.
    sinon
      Parcourir les pages liées à la case $h(c)$ jusqu'à en trouver une qui posséde une place suffisante pour le nouvel enregistrement.
      si il en existe une **alors**
        Y insérer l'enregistrement : l'insertion est terminée.
      sinon (le bout de la chaîne est atteint)
        Créer une nouvelle page.
        L'ajouter au bout de la chaîne des pages liées à la case $h(c)$.
        Y insérer l'enregistrement : l'insertion est terminée.
      fsi
    fsi
  fin

Suppression d'un enregistrement

- (Il s'agit de supprimer l'enregistrement de clé *c*.)
  début
    Rechercher l'enregistrement de clé *c*.
    si il n'existe pas **alors**
      La suppression est terminée.
    sinon
      Supprimer cet enregistrement de la page qui le contient.
      si cette page devient vide **alors**
        La supprimer et remettre à jour la chaîne des pages.
        La suppression est terminée.
      fsi
    fsi
  fin

L'index `dico` par hachage sans répertoire

On peut éviter l'utilisation d'un répertoire en créant un index de *N* pages contiguës.
Le code haché donne alors directement accès à la page contenant les clés recherchées.



Performances

- Sans répertoire et avec un bon ajustement des valeurs de *N* et du nombre d'enregistrements par page, on peut accéder à un enregistrement en 1,1 ou 1,2 accès disque en moyenne.
- On peut obtenir la même performance avec un répertoire pourvu que celui-ci tienne en mémoire centrale.
- Le répertoire permet de ne pas associer inutilement une page à un code-haché non instancié (2 dans notre exemple). Par contre, un répertoire occupe de la place et s'il ne tient pas en mémoire centrale le nombre d'accès disque sera augmenté.
- Le hachage statique est mal adapté à un fichier de données très évolutif car la taille du répertoire peut s'avérer sous-évaluée, entraînant un accroissement du nombre de pages de débordement et donc du nombre d'accès disque.
- Pour y pallier, des méthodes de hachage dites dynamiques ont été proposées, qui font évoluer la fonction de hachage en fonction du nombre d'enregistrements.

Hachage dynamique

- L'idée est la suivante : lorsqu'une page est saturée, au lieu de créer une page de débordement, on fait évoluer la fonction de hachage, afin d'assurer que le recherche d'un enregistrement ne nécessitera qu'un seul accès à une page.
- Deux méthodes ont été proposées :
  - le **hachage extensible**, par Fagin et al. en 1979,
  - le **hachage linéaire** par W. Litwin en 1980.

Hachage extensible

- Une suite de bits est associée à chaque clé.
- L'évolution de la clé de hachage consiste à augmenter le nombre de bits à prendre en compte dans une clé pour trouver la case du répertoire qui pointe vers la page contenant l'enregistrement associé à cette clé.
  - si tous les enregistrements tiennent dans une seule page, 0 bits seront à prendre en compte,
  - s'ils occupent 2 pages, 1 bit sera à prendre en compte,
  - s'ils occupent 3 ou 4 pages, 2 bits seront à prendre en compte,

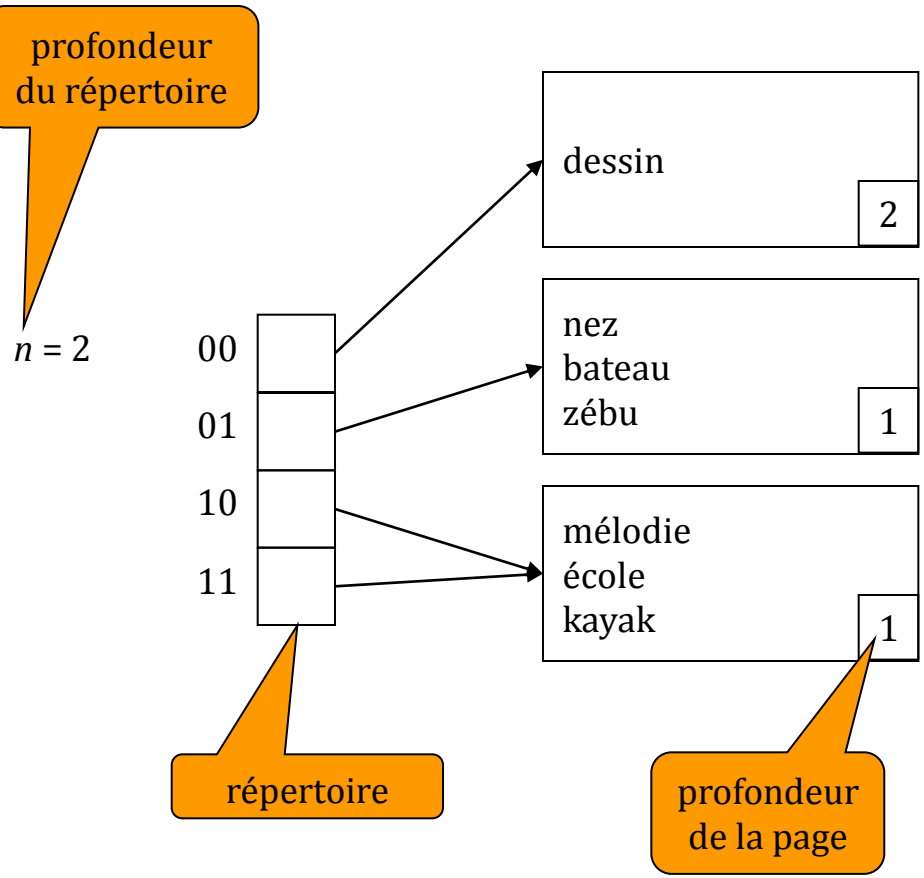
  ...

Organisation

- La fonction de hachage h associe à chaque clé une séquence suffisamment longue de bits (32, par exemple).
- L'index est composé de deux parties :
  - un ensemble de pages.
  - un répertoire de 2^n cases ($n \geq 0$, profondeur du répertoire) dont chacune contient un pointeur vers une page.
- Plusieurs cases consécutives du répertoire peuvent pointer vers la même page.
- Un enregistrement de clé c est stocké dans la page pointée par la i^e case du répertoire telle que i est égal au nombre formé par les n premiers bits de $h(c)$.
- A chaque page est associé un nombre entier m ($0 \leq m \leq n$), appelée profondeur de la page.
- Si une page a la profondeur m, alors il y a 2^{n-m} cases du répertoire qui pointent vers elle.

L'index `dico` par hachage extensible

c	*h*(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100



Insertion d'un enregistrement (1)

- (Il s'agit d'insérer un enregistrement *e* de clé *c*)
  début
  Rechercher la page P qui devrait contenir e : soit m sa profondeur.
  si P contient e **alors**
  L'insertion est terminée.
  sinon
  si P n'est pas saturée **alors**
  Insérer *e* dans *P* : l'insertion est terminée.
  sinon
  si $m < n$ **alors** (partage d'une page en 2)
  Créer une nouvelle page P'.
  $m = m + 1$
  profondeur locale de *P* = profondeur locale de *P'* = *m*.
  Enregistrer dans *P'* tous les enregistrements de *P*
  dont le m^e bit est égal à 1.
  Faire pointer vers *P'* chaque case du répertoire qui pointait vers *P*,
  si son m^e bit est égal à 1.
  Recommencer l'insertion de *e*.
  ...

Insertion d'un enregistrement (2)

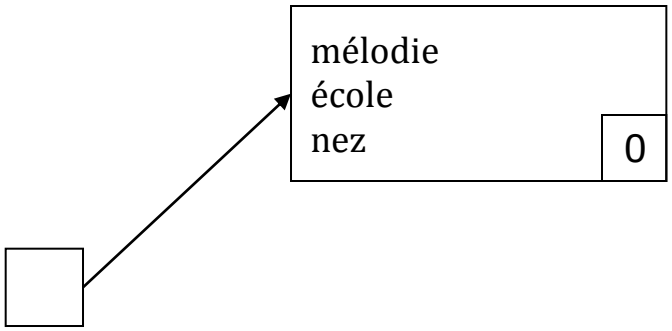
```
...
                sinon (doublement du répertoire)
                    Doubler la profondeur du répertoire
                    en dédoublant chaque case du répertoire.
                    n = n + 1.
                    Recommencer l'insertion de e.
                fsi
            fsi
        fsi
fin
```

Construction de l'index `dico` (1)

Insertions successives des enregistrements : mélodie, école, nez

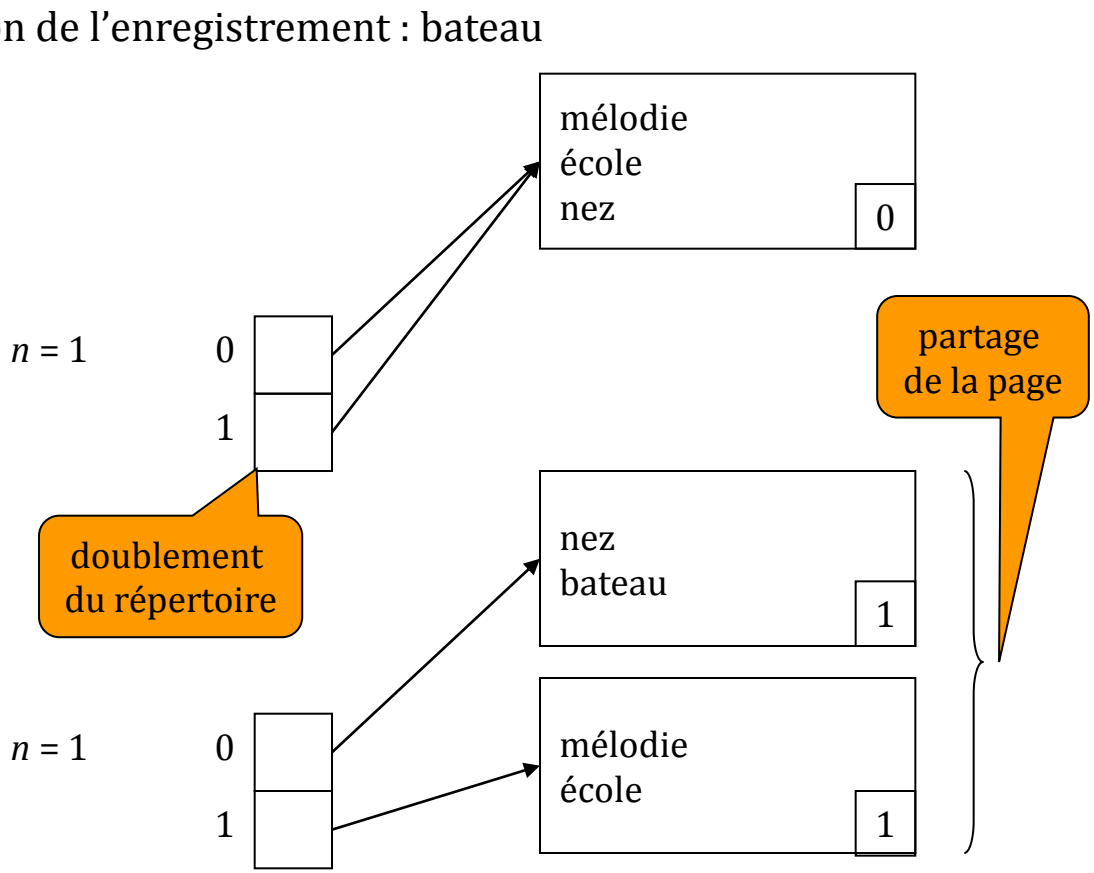
c	***h*(c)**
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101

$n = 0$



Construction de l'index `dico` (2)

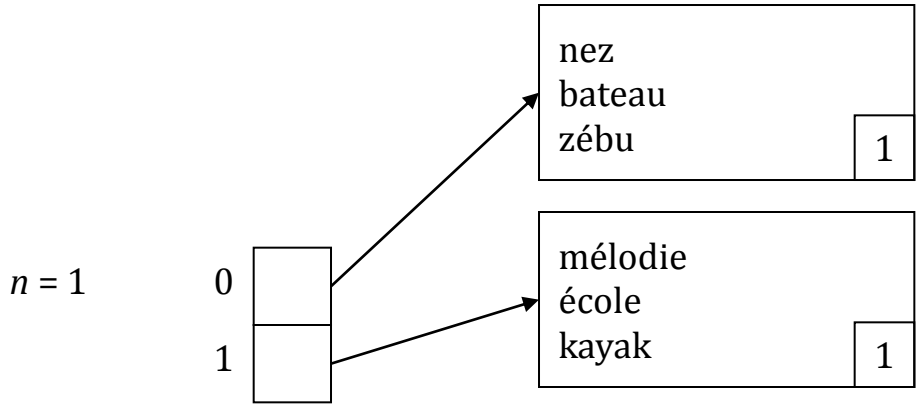
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index `dico` (3)

Insertions successives des enregistrements : kayak, zébu

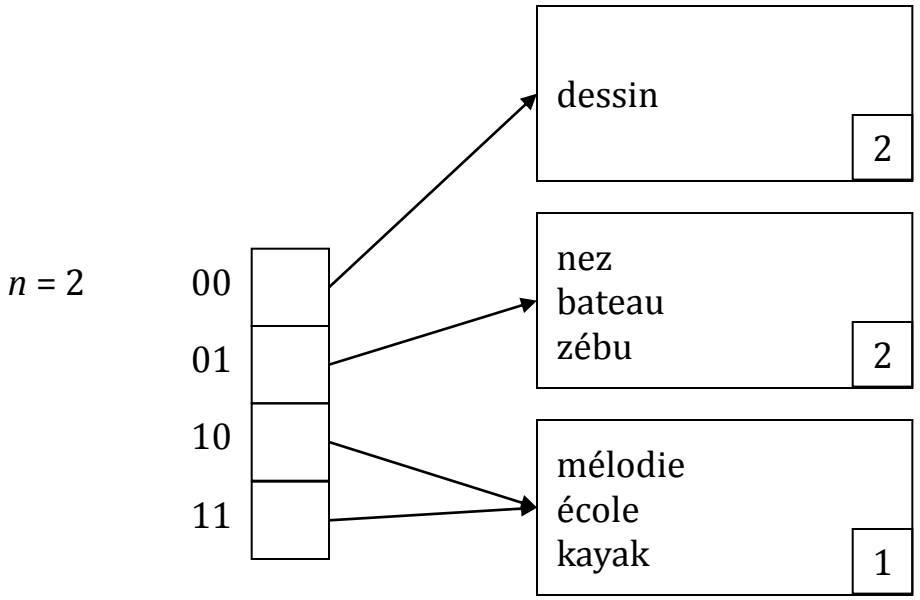
c	***h*(c)**
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index `dico` (4)

Insertions successives des enregistrements : dessin

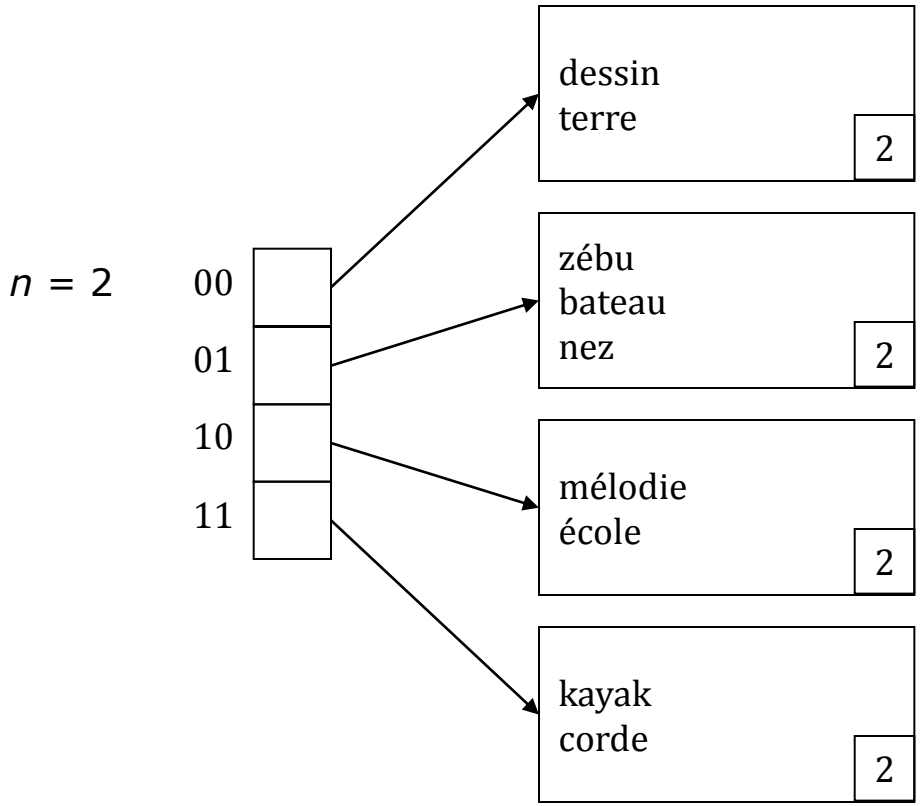
c	***h*(c)**
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index `dico` (5)

Insertions successives des enregistrements : corde, terre

c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Résolution des requêtes

Introduction

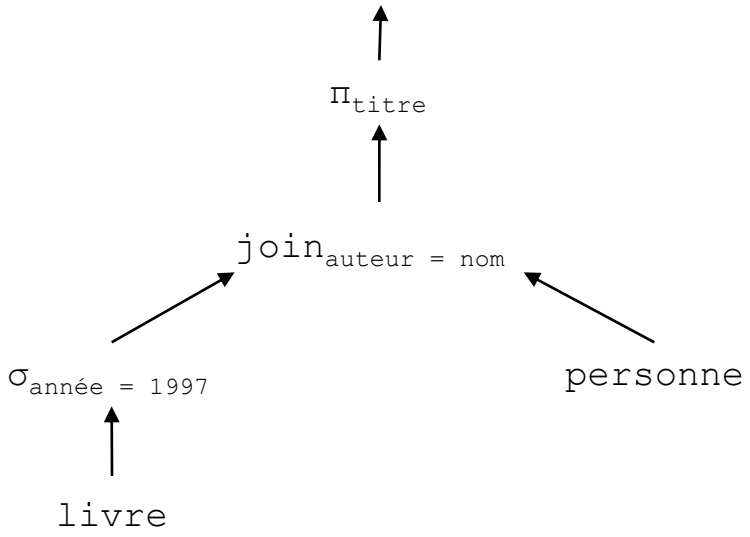
- L'accès à une BD relationnelle est réalisé au travers d'un langage de requêtes (SQL, le plus souvent). Les performances de l'évaluateur de requêtes sont donc cruciales.
- Un langage de requêtes est déclaratif : l'utilisateur exprime sa requête mais pas la façon d'y répondre.
- Or pour une requête donnée, il existe en général plusieurs stratégies pour construire la réponse. C'est au SGBD de choisir la stratégie optimale.

Les 3 phases de la résolution d'une requête

- La résolution d'une requête à une BD relationnelle se déroule en trois phases :
  1. **Traduction** de la requête en un arbre d'opérateurs relationnels (sélection, jointure, projection, etc.) : l'**arbre de requête.**
  2. **Optimisation.** Il s'agit de déterminer un plan d'exécution de coût minimal. Deux procédures sont principalement mises en œuvre :
     - **transformation de l'arbre** de requête en un arbre équivalent, basée sur les propriétés d'associativité et de commutativité de l'algèbre relationnelle ;
     - **évaluation du coût** de résolution de chaque opérateur.
  3. **Évaluation** du plan d'exécution produit par la phase 2.

Arbre de requête

```
SELECT titre
  FROM livre, personne
 WHERE année = 1997
   AND auteur = nom
```



Evaluation des opérateurs relationnels

- Nous étudierons quelques méthodes d'évaluation des opérateurs :
  - de sélection,
  - de jointure,
  - de projection.

Méthodes d'évaluation d'une sélection

- **Simple boucle** (parcours séquentiel)
- **Indexée**
  - Utilisation d'un index primaire pour retrouver un *n*-uplet dont la clé est donnée ou appartient à un intervalle donné,
  - Utilisation d'un index secondaire pour retrouver un ensemble de *n*-uplets dont la clé est donnée ou appartient à un intervalle donné.

Méthodes d'évaluation d'une jointure

- Sur une condition quelconque :
  - **boucles imbriquées**
- Equi-jointure :
  - pas d'index sur les constituants de jointure :
    - **boucles imbriquées**
    - tri-fusion
    - **hachage**
  - index sur l'un des constituants de jointure :
    - **indexée**
  - index triés sur les deux constituants de la jointure :
    - fusion des feuilles des deux index

Méthodes d'évaluation d'une projection

- Simple boucle
- Tri puis simple boucle, si les doubles doivent être éliminés

Étude de quelques méthodes

- On supposera que chaque relation est stockée dans un fichier qui contient les *n*-uplets de cette relation.
- Le coût d'une opération sera mesuré en nombre de transferts de pages entre disque et mémoire centrale.
- Ce coût peut se décomposer en deux parties :
  - un **coût de production** de la relation résultat,
  - un **coût d'écriture** de cette relation.
- Le coût de production dépend de la méthode choisie pour réaliser l'opération alors que le coût d'écriture en est indépendant.
- Dans le cas d'une évaluation pipeline d'une suite d'opérations :

  $op_1, ..., op_n$

  les résultats des opérations $op_1, ..., op_{n-1}$ ne sont pas stockés sur disque car les *n*-uplets produits sont directement transmis à l'opérateur suivant :
  - les coûts de lecture d'une relation opérande ou d'écriture des résultats d'un opérateur peuvent donc être nuls.

Coût de production d'une opération

- Le coût de production de la relation résultat dépend :
  - de la méthode utilisée pour résoudre l'opération,
  - des paramètres d'implantation des relations opérandes.
- Le jeu de paramètres utilisé sera le suivant :

$card(R)$	cardinalité de la relation R
$nb(R)$	nombre de blocs occupés par le fichier R
$nbatt(R)$	nombre d'attributs de la relation R
$nv(R, X)$	nombre de valeurs différentes du constituant X de la relation R

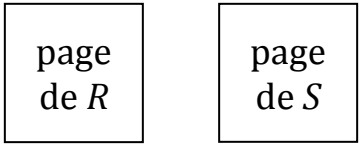
Coût d'écriture du résultat

- Le coût d'écriture de la relation résultat est indépendant de la méthode de résolution utilisée.
- Cette écriture est réalisée au travers du tampon de la façon suivante :
  - Une case *C* du tampon est affectée à la relation résultat initialisée avec une page vide *P*.
  - Pour chaque *n*-uplet *t* produit :
    - Si la page *P* est pleine, l'écrire sur un bloc du disque.
    - Placer une nouvelle page vide *P* dans la case *C*.
    - Ecrire *t* dans *P*.
- Le coût d'écriture du résultat est donc égal à *nb*(*relation résultat*).

Sélection par simple boucle

$$S := \sigma_{cond}(R)$$

Tampon



Algorithme

Pour chaque bloc du fichier *R* **faire**
    Transférer la page contenue dans ce bloc, dans la case 1.
    Pour chaque *n*-uplet *t* de cette page **faire**
        si *cond*(*t*) **alors** écrire *t* dans la page de la case 2
        ou le transmettre à l'opérateur suivant.

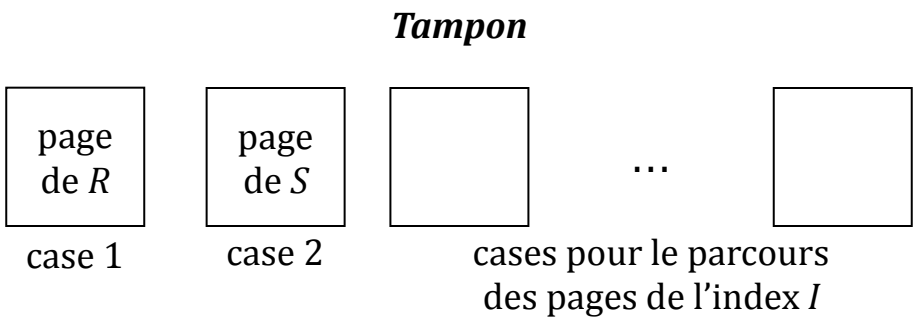
Coût de production

$nb(R)$

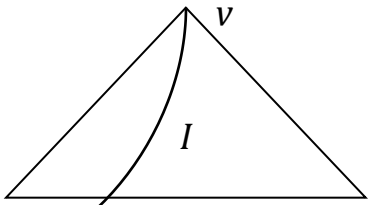
Sélection indexée (1)

$$S := \sigma_{A = v}(R)$$

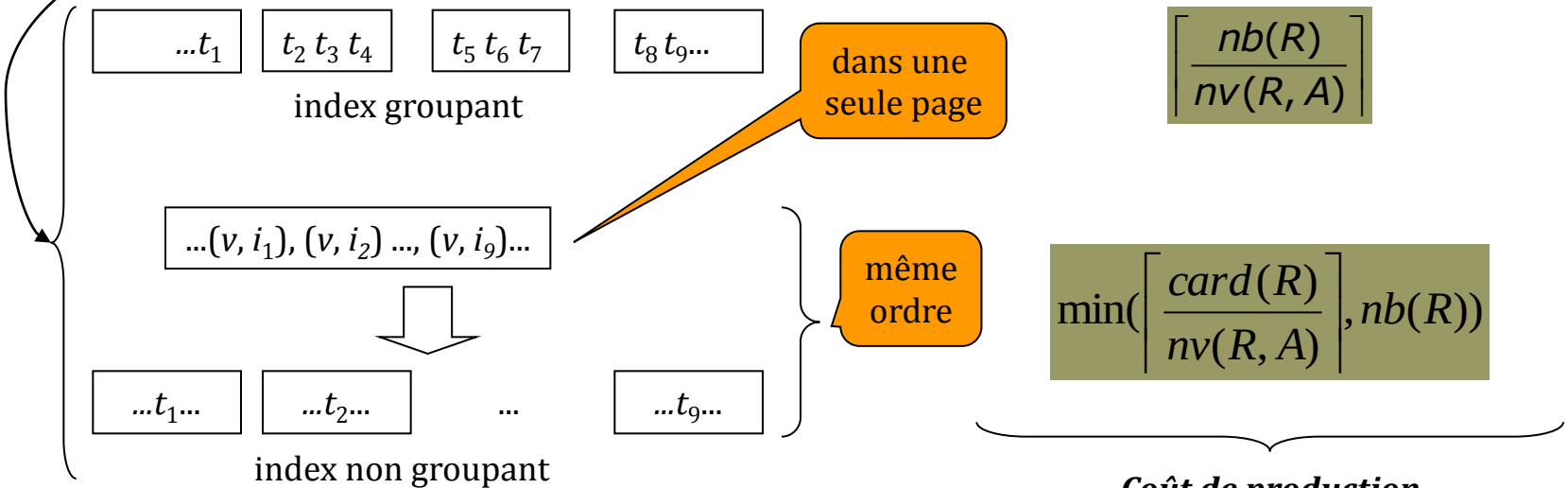
A est un attribut de R
Il existe un index I sur l'attribut A de R



Sélection indexée (2)



I = index de l'attribut A de le relation R



$$\left\lceil \frac{nb(R)}{nv(R,A)} \right\rceil$$

$$\min\left(\left\lceil \frac{card(R)}{nv(R,A)} \right\rceil, nb(R) \right)$$

Coût de production
(négligeant le parcours de l'index)

Comparaison des deux méthodes

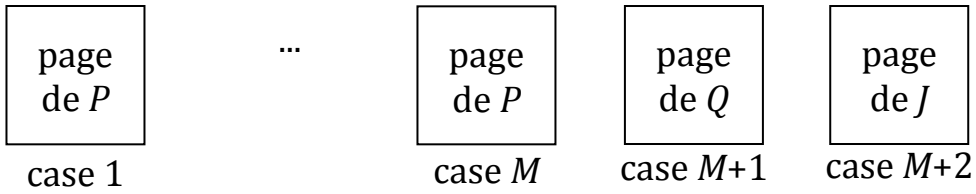
- Soit une relation `livre` telle que :
  - *card*(`livre`) = 12000
  - *nb*(`livre`) = 2400
  - *nv*(`livre`, `année`) = 50
  - Il existe un index *I* sur l'attribut année
- Le coût de production de l'opération :
  - $\sigma_{\texttt{année = 2000}}(\texttt{livre})$

  est égal à :
  - sélection par boucle :
    - 2400
  - sélection indexée :
    - $\lceil 2400/50 \rceil = 48$, si l'index est groupant,
    - $\min(\lceil 12000/50 \rceil, 2400) = 240$, si l'index est non groupant.

Jointure par boucles imbriquées (1)

J := P join$_{cond}$ Q

Tampon



Algorithme

pour chaque suite de M blocs de P **faire**
Transférer les pages contenues dans ces blocs dans les cases 1 à M.
pour chaque chaque bloc de Q **faire**
Transférer la page contenue dans ce bloc dans la case M + 1.
pour chaque n-uplet t' de cette page **faire**
pour chaque page contenue dans les cases 1 à M
et **pour chaque** n-uplet t de cette page **faire**
si $cond(t\ conc\ t')$ est vraie **alors**
Ecrire le n-uplet $t\ conc\ t'$ dans la page de la case M + 2
ou le transmettre à l'opérateur suivant.

Jointure par boucles imbriquées (2)

Coût de production

$$nb(P) + \left\lceil \frac{nb(P)}{M} \right\rceil \times nb(Q)$$

Si $nb(P) \leq M$, c.-à-d. si la relation P tient en mémoire, le coût est égal à :

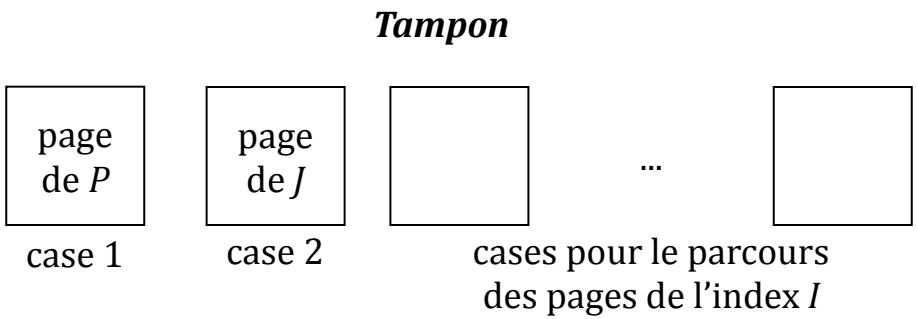
$$nb(P) + nb(Q)$$

Le coût est minimum si l'on choisit la relation de cardinalité minimum comme relation externe.

Jointure indexée (1)

$$J := P \ \texttt{join}_{A = B} \ Q$$

A est un attribut de P
B est une attribut de Q
Il existe un index *I* sur l'attribut *B* de *Q*



Jointure indexée (2)

Algorithme

pour chaque bloc de *P* **faire**
    Transférer dans la case 1 la page *p* contenue dans ce bloc.
    pour chaque *n*-uplet *t* de *p* **faire**
        v = *t.A*
        Effectuer la sélection indexée $\sigma_{B=v}(Q)$ (index *I*)
        pour chaque *n*-uplet *t'* sélectionné par cette opération **faire**
            Ecrire le *n*-uplet *t conc t'* dans la page de la case 2 ou le transmettre à l'opérateur suivant.

Coût de production

$$nb(P) + card(P) \times \left\lceil \frac{nb(Q)}{nv(Q,B)} \right\rceil$$ si l'index *I* est groupant

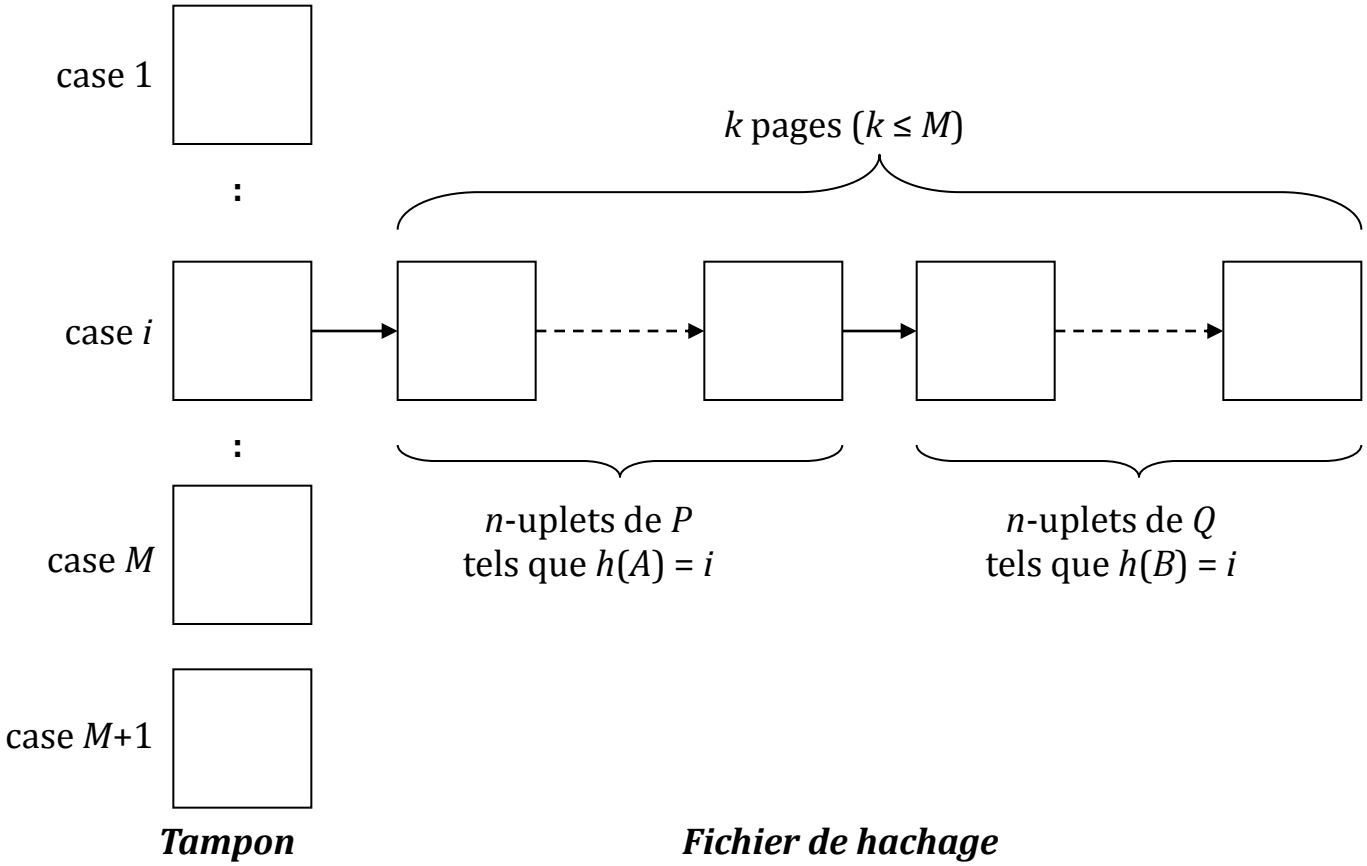
$$nb(P) + card(P) \times \min\left(\left\lceil \frac{card(Q)}{nv(Q,B)} \right\rceil, nb(Q) \right)$$ si l'index *I* est non groupant

Équi-jointure par hachage (1)

$$J := P \texttt{ join}_{A = B} Q$$

A est un attribut de P
B est une attribut de Q
h est une fonction de hachage sur les valeurs de A et B

Équi-jointure par hachage (2)



Équi-jointure par hachage (3)

Algorithme

pour chaque bloc de *P* **faire**
    Transférer dans la case *M* + 1 la page contenue dans ce bloc.
    Hacher ses *n*-uplets.
pour chaque bloc de *Q* **faire**
    Transférer dans la case *M* + 1 la page contenue dans ce bloc.
    Hacher ses *n*-uplets.
Pour chaque chaîne de pages liée à la case *i* du tampon **faire** :
    Charger les pages de cette chaîne dans les *M* cases du tampon.
    Effectuer la jointure dans la case *M* + 1.

Condition

Il faut que le nombre de pages liées à une case du répertoire soit ≤ M pour qu'elles tiennent dans le tampon.
Si on suppose que la répartition des *n*-uplets est uniforme, on doit avoir :

$$nb(P) + nb(Q) \leq M^2$$

Coût de production

$$3 \times (nb(P) + nb(Q))$$

Comparaison des trois méthodes

- Soit une relation `livre(ISBN, nom_auteur)` telle que :
  - *card*(`livre`) = 24000, *nb*(`livre`) = 2400,
  - *nv*(`livre`, `nom_auteur`) = 6000,
  - Il existe un index `I` non groupé sur l'attribut `nom_auteur`.

  et une relation `auteur(nom, pays)` telle que :
  - *card*(`auteur`) = 6000, *nb*(`auteur`) = 600
  - *nv*(`auteur`, `nom`) = 6000
- 102 cases sont disponibles dans le tampon.
- Le coût de production de l'opération :
  - $\texttt{auteur}\ \texttt{join}_{\texttt{nom = nom\_auteur}}\ \texttt{livre}$

  est égal à :
  - jointure par boucles imbriquées :
    - $600 + \lceil 600 / 100 \rceil \times 2400 = 15000$
  - jointure indexée :
    - $600 + 6000 \times \min(\lceil 24000 / 6000 \rceil,\ 2400) = 24600$
  - jointure par hachage :
    - $3 \times (2400 + 600) = 9000$

    qui est donc ce cas, la meilleure solution.

Projection sans élimination des doublons

$$P := \Pi_{A1, \ldots, Ak}(R)$$

Coût de production

$nb(R)$

Évaluation de la taille de la relation résultat

- La taille de la relation produite par un opérateur est mesurée en nombre de *n*-uplets et en nombre de blocs.
- Cette évaluation n'est pas toujours simple. Elle nécessite de disposer de connaissances statistiques sur les données.
- Nous ne traitons ici que les cas les plus simples.
- On suppose que :
  - les valeurs d'un attribut sont réparties uniformément sur les *n*-uplets d'une relation,
  - tous les attributs d'une même relation ont la même taille.

Taille du résultat d'une sélection

$$card(\sigma_{A=v}(R)) = \left\lceil \frac{card(R)}{nv(R,A)} \right\rceil$$

$$nb(\sigma_{A=v}(R)) = \left\lceil \frac{nb(R)}{nv(R,A)} \right\rceil$$

$$card(\sigma_{c1 \wedge c2}(R)) = card(\sigma_{c2}(\sigma_{c1}(R)))$$

$$nb(\sigma_{c1 \wedge c2}(R)) = nb(\sigma_{c2}(\sigma_{c1}(R)))$$

$$card(\sigma_{A1=v1 \wedge A2=v2}(R)) = \left\lceil \frac{card(R)}{nv(R,A_1) \times nv(R,A_2)} \right\rceil$$

$$nb(\sigma_{A1=v1 \wedge A2=v2}(R)) = \left\lceil \frac{nb(R)}{nv(R,A_1) \times nv(R,A_2)} \right\rceil$$

Taille du résultat d'une jointure

Dans le cas où $nv(P, A) \leq nv(Q, B)$ on peut faire l'hypothèse que chaque n-uplet de P se joint avec au moins un n-uplet de Q et inversement.

$$card(P\ join_{A=B}\ Q) = \left\lceil \frac{card(P) \times card(Q)}{\max(nv(P, A), nv(Q, B))} \right\rceil$$

$$nb(P\ join_{A=B}\ Q) = \left\lceil \frac{card(P) \times nb(Q) + card(Q) \times nb(P)}{\max(nv(P, A), nv(Q, B))} \right\rceil$$

Taille du résultat d'une projection

On suppose que les valeurs des attributs d'une relation ont la même taille (ce qui est très approximatif).

$$card(\pi_{A1,\ldots,Ak}(R)) = card(R)$$

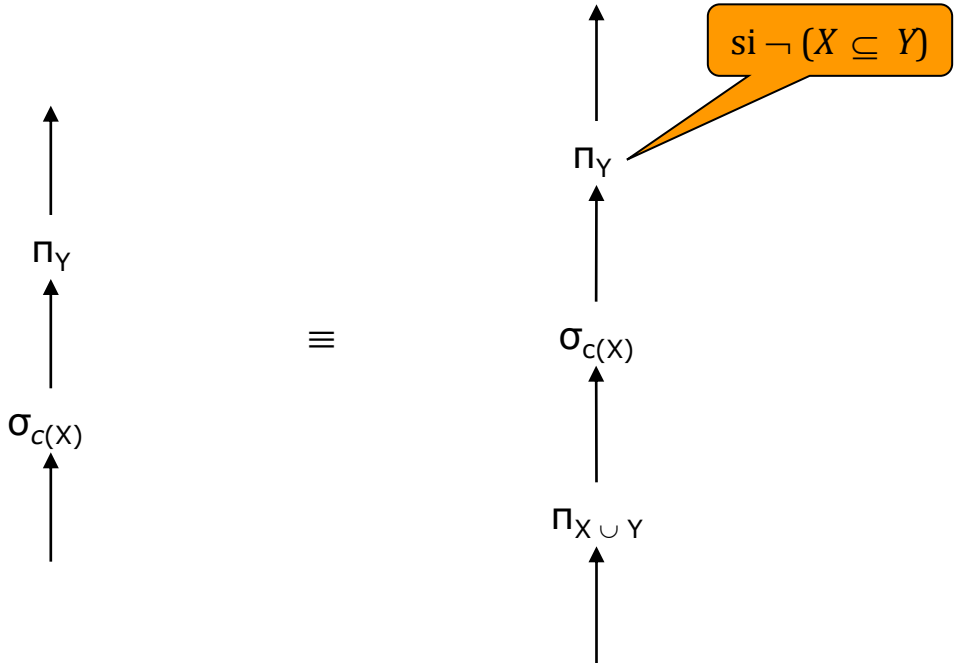
$$nb(\pi_{A1,\ldots,Ak}(R)) = \left\lceil \frac{k}{nbatt(R)} \times nb(R) \right\rceil$$

Propriétés des opérateurs relationnels (1)

T1 : Eclatement et regroupement des sélections

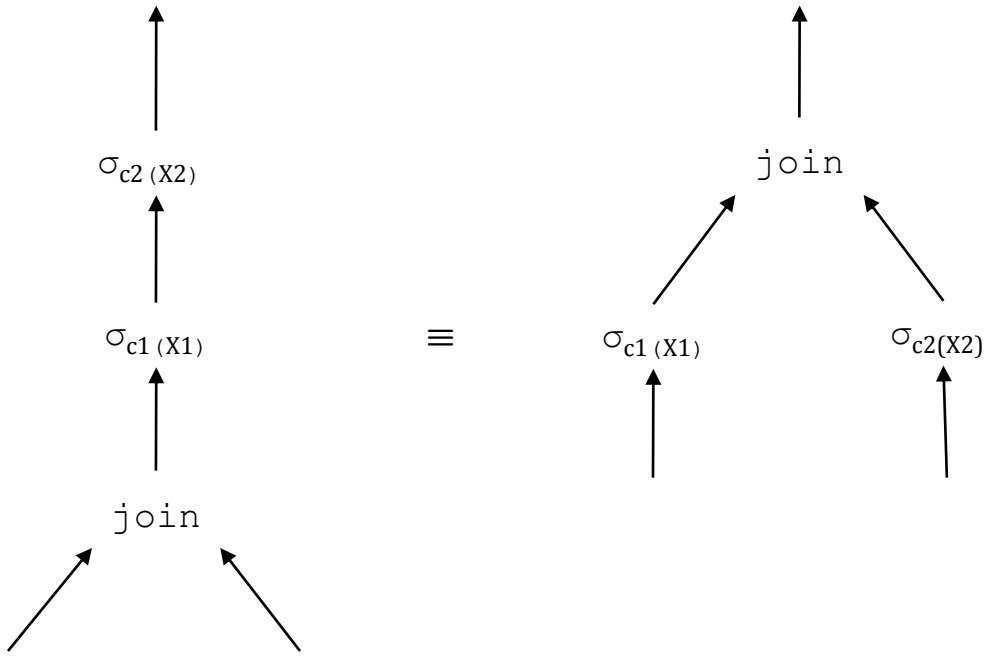
Propriétés des opérateurs relationnels (2)

T2 : Permutation projection-sélection



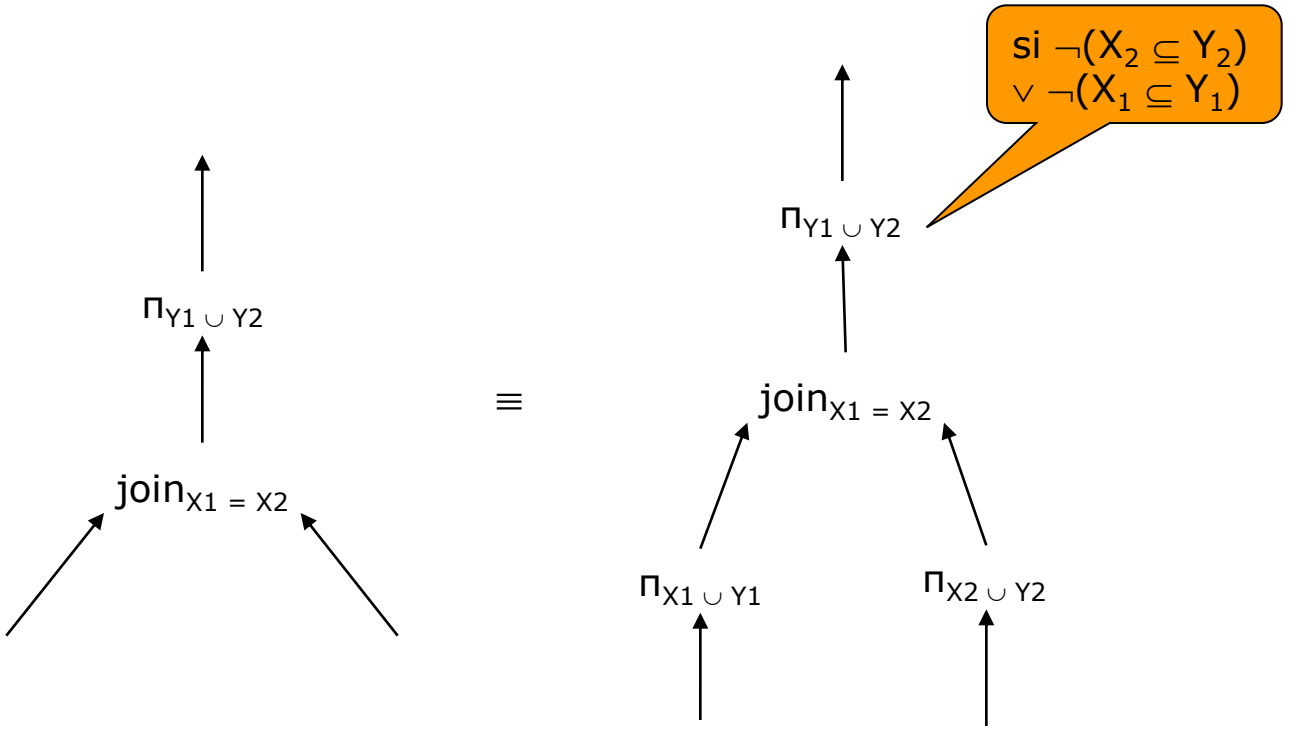
Propriétés des opérateurs relationnels (3)

T3 : Permutation sélection-jointure



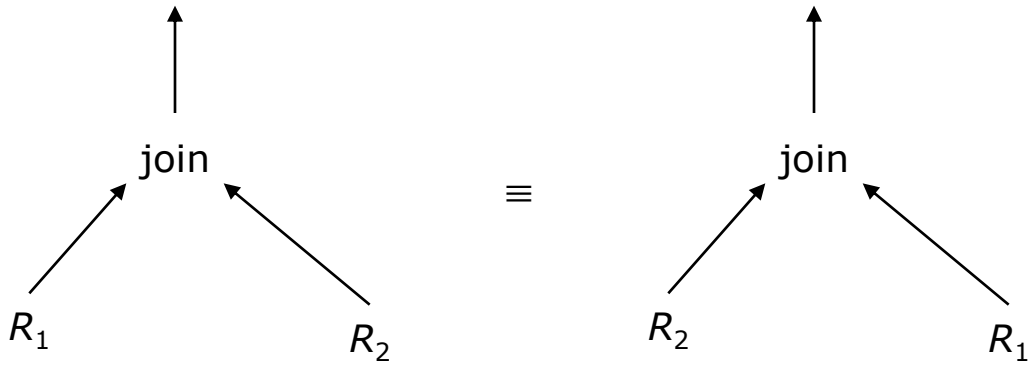
Propriétés des opérateurs relationnels (4)

T4 : Permutation projection-jointure



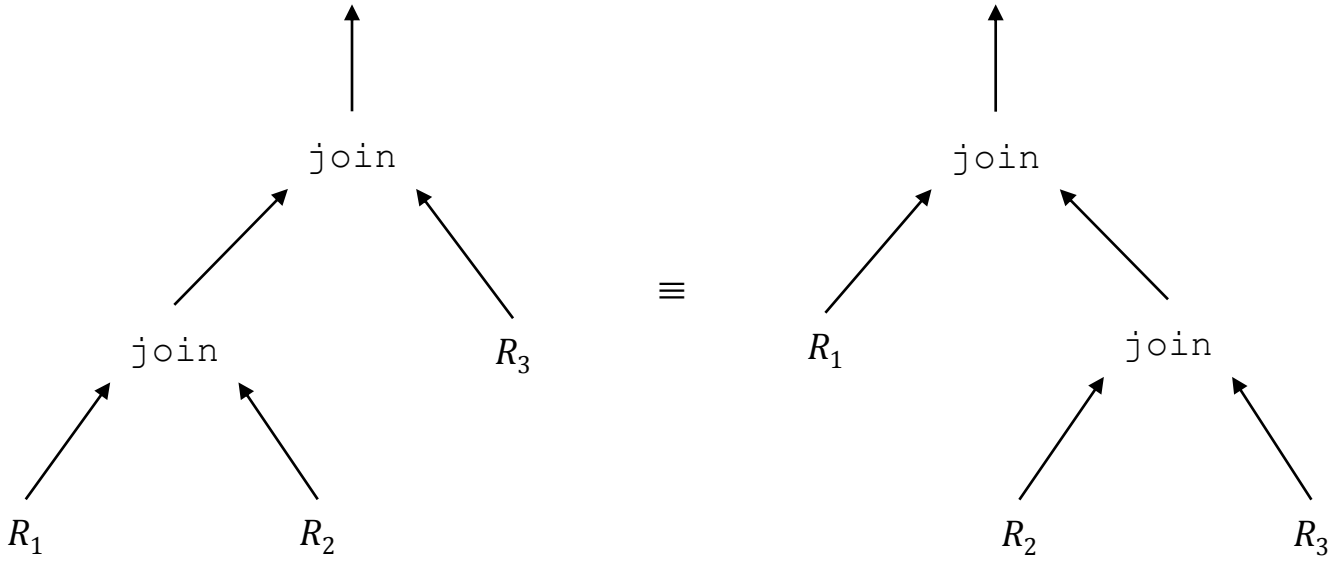
Propriétés des opérateurs relationnels (5)

T5 : Commutativité de la jointure



Propriétés des opérateurs relationnels (6)

T6 : Associativité de la jointure



Choix d'un plan d'exécution (1)

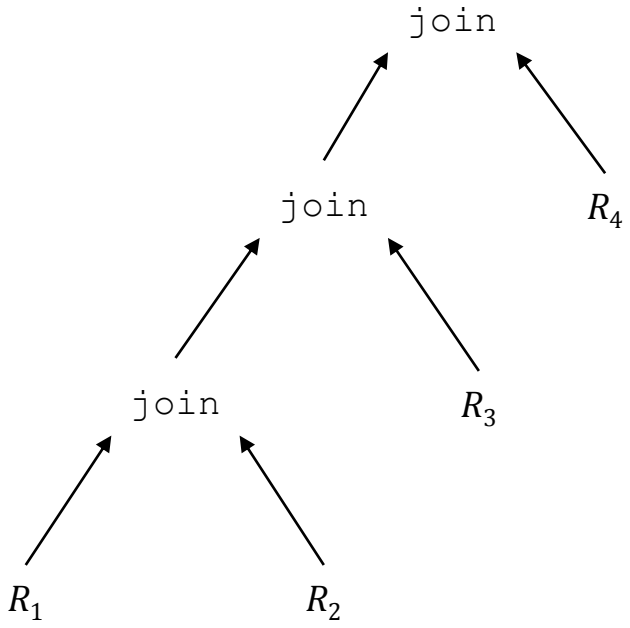
- En théorie, le choix d'un plan d'exécution peut être effectué de la façon suivante :
  - Traduire la requête sous forme d'un arbre de requête initial :
    Dans le cas où la requête est un bloc `SELECT...FROM...WHERE...`, l'arbre de requête initial est obtenu en réalisant la jointure ou le produit cartésien des relations impliquées (clause `FROM`), suivi d'une sélection (clause `WHERE`) suivie d'une projection (clause `SELECT`).
  - Construire l'ensemble des arbres équivalents à l'arbre initial en appliquant les propriétés de commutativité et d'associativité des opérateurs.
  - A partir de chacun de ces arbres générer tous les plans d'exécution possibles en associant à chaque opérateur chacune des méthodes de résolution applicable.
  - Évaluer le coût de chaque plan d'exécution.
  - Choisir le plan de coût minimal.

Choix d'un plan d'exécution (2)

- L'inconvénient majeur de cette méthode est que le nombre de plans d'exécution à examiner croit très vite avec le nombre de relations sur lesquelles porte la requête.
- Par exemple, on peut montrer que le nombre d'ordres de jointures de n relations est :
  - $(2 \times (n-1))\,!\,/\,(n-1)\,!$
- Ce nombre croit donc très vite quand n augmente. Il est égal à 12 pour $n = 3$, 1680 pour $n = 5$, 665280 pour $n = 7$, ...
- En pratique, on utilise des règles heuristiques pour diminuer le nombre de plans d'exécution à évaluer.

Arbres linéaires de jointure

- Pour minimiser le nombre d'ordres de jointure à considérer on se limite en général aux arbres de requêtes dits **arbres linéaires gauches de jointure** où les sous-arbres droits sont des arbres de requête sans opérateurs de jointure.
- Pour *n* relations il y a *n*! arbres linéaires gauche de jointure. Ce qui est beaucoup moins que le nombre total d'ordres de jointure.
- Par exemple, il y 1680 ordres de jointure de 5 relations dont 120 correspondent à des arbres de jointure linéaires gauches.
- Un arbre linéaire gauche de jointure est particulièrement intéressant lorsqu'il existe un index sur l'attribut de jointure ou en cas de résolution *pipeline*.



Etude de cas

- Schéma
  - `livre(isbn, titre, auteurs, nom_éditeur, année, prix)`
  - `éditeur(nom, adresse, pays)`
- Requête
  - *Titres des livres publiés en 2000 édités par un éditeur espagnol ?*

```
SELECT titre
FROM livre, editeur
WHERE nom = nom_editeur
  AND pays = 'Espagne'
  AND année = 2000;
```

Paramètres

Tables

	nbatt	*card*	*nb*	*nv*(`titre`)	*nv*(`nom_éditeur`)	*nv*(`année`)
`livre`	6	12000	2400	12000	300	20

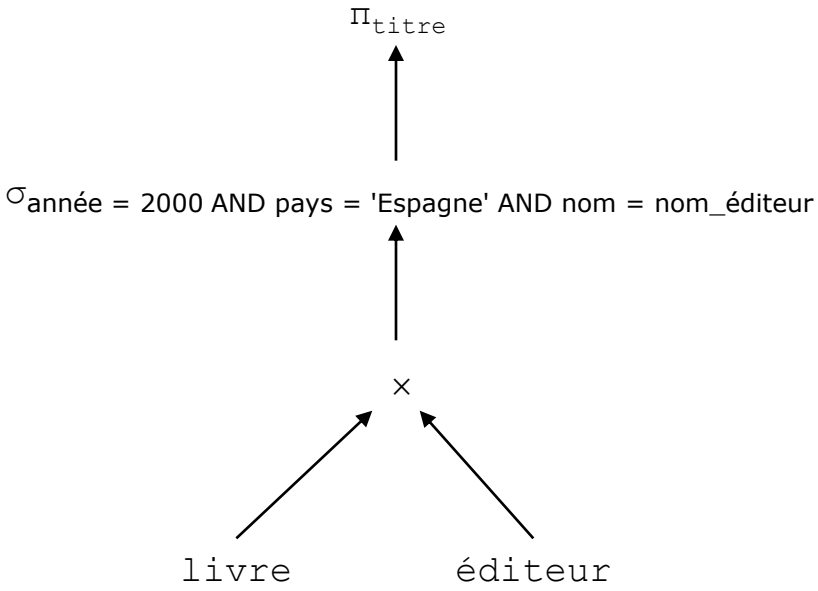
	nbatt	*card*	*nb*	*nv*(`nom`)	*nv*(`pays`)
`éditeur`	3	300	60	300	30

Index

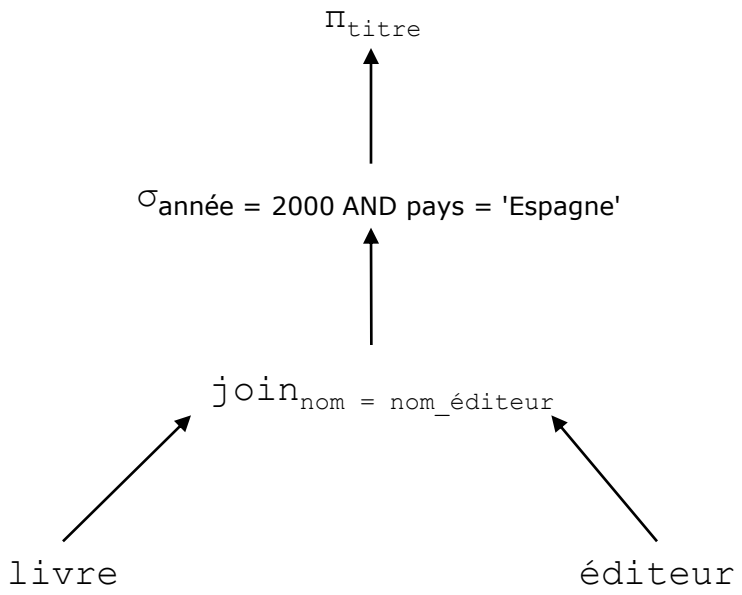
	index groupant ?
`livre(isbn)`	oui
`livre(nom_éditeur)`	non
`livre(année)`	non
`éditeur(nom)`	oui
`éditeur(pays)`	non

Tampon
52 cases

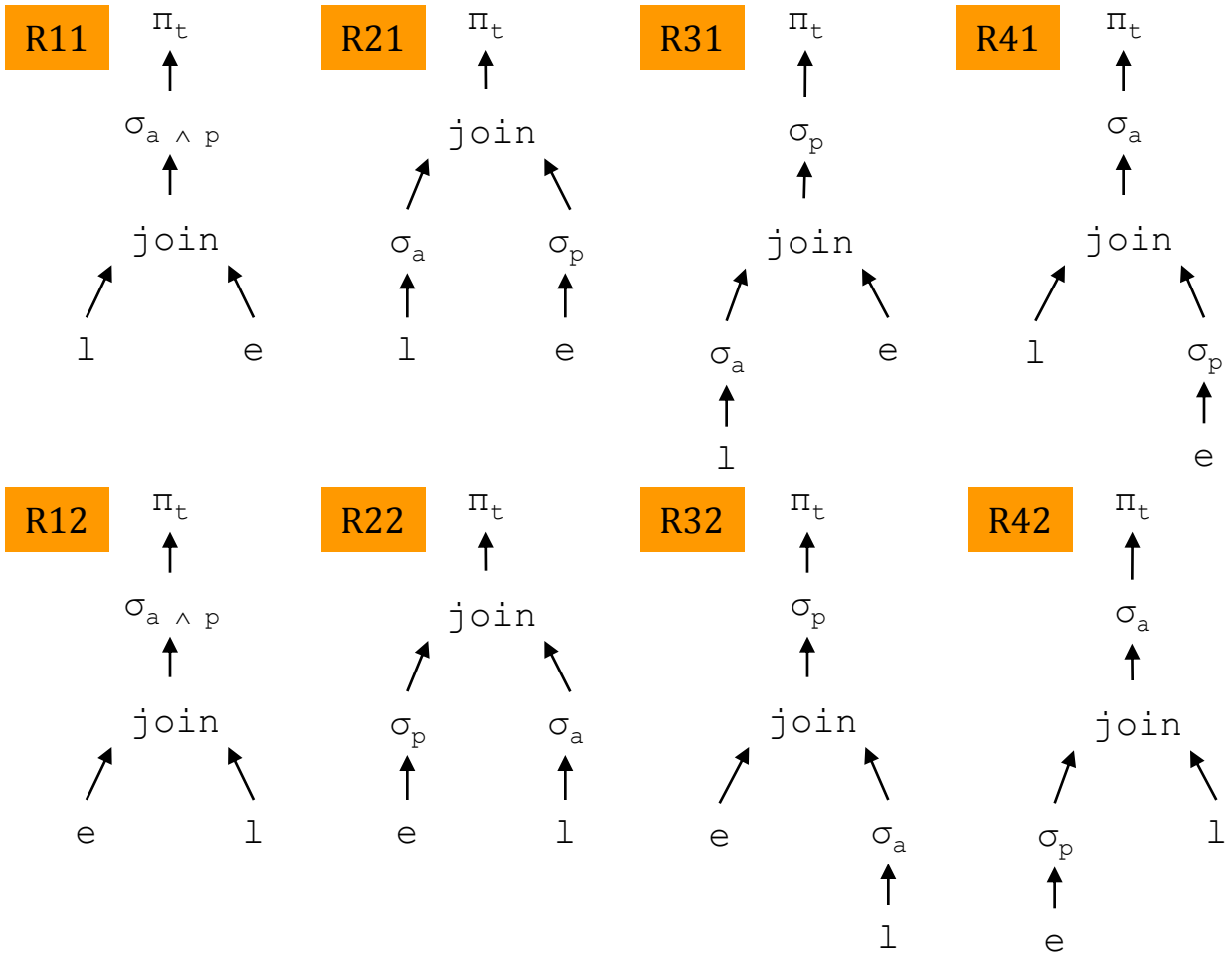
Arbre de requêtes initial



Arbre de requêtes avec jointures



Arbres de requêtes équivalents



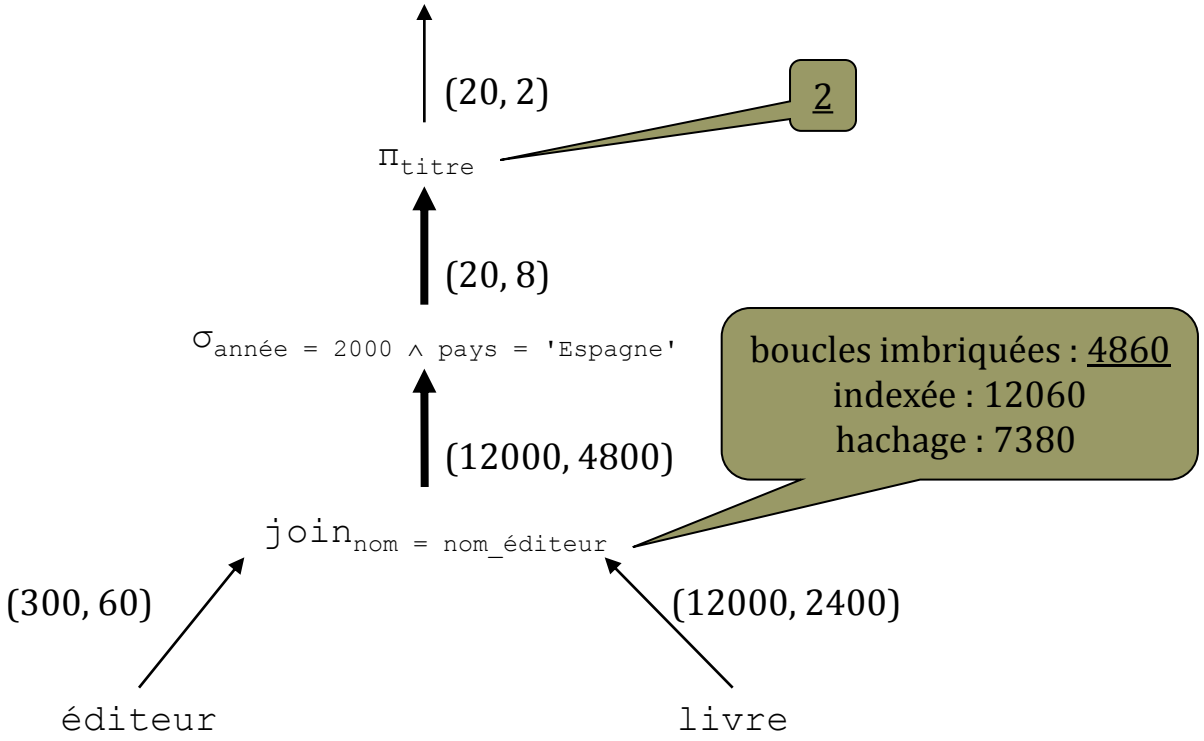
Par application des règles T1, T3 et T5 à l'arbre initial R11

```
l = livre
e = éditeur
a = année = 2000
p = pays = 'Espagne'
```

Étude de 4 plans d'exécution

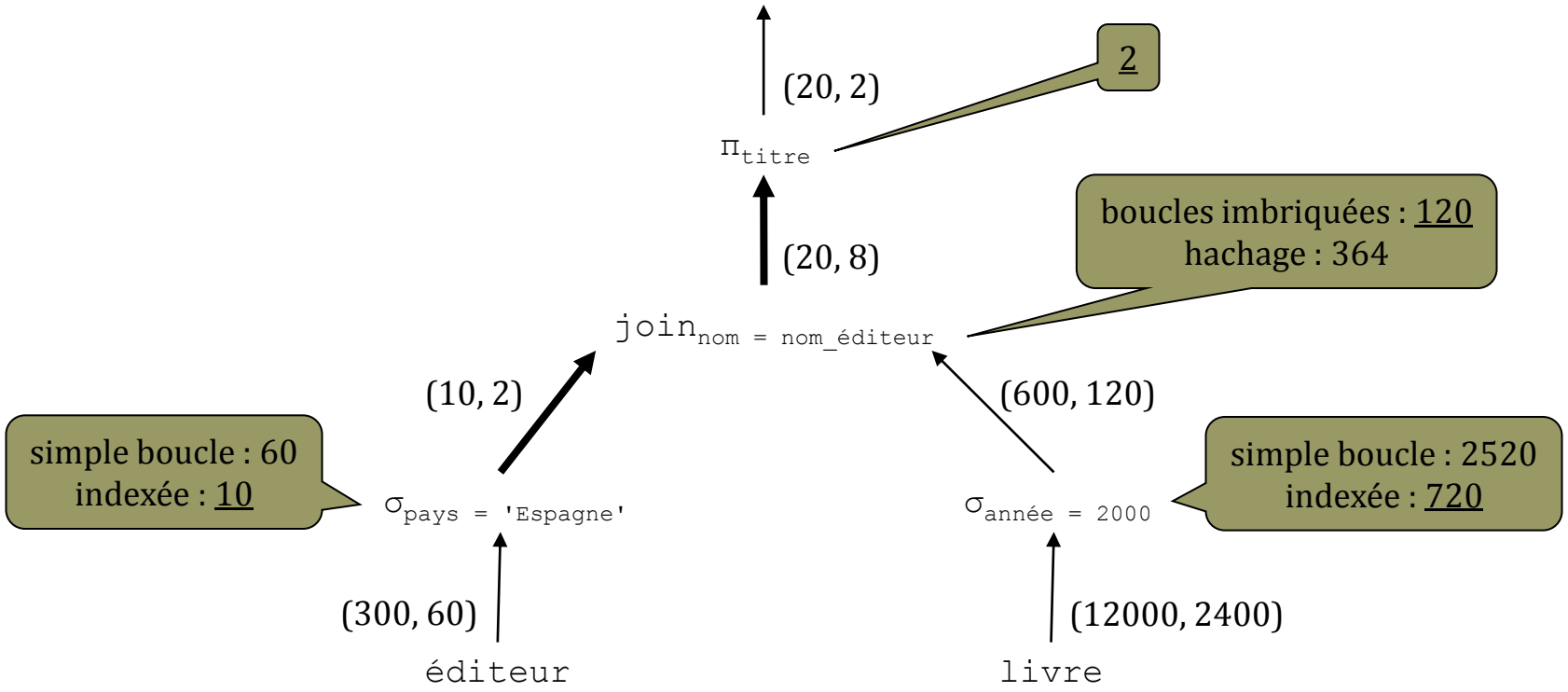
- On évalue les plans d'exécution correspondant aux arbres de requête R12, R22, R32 et R42.
- Ces plans consistent à réaliser la jointure en choisissant comme relation externe la relation `éditeur` qui est beaucoup moins volumineuse que la relation `livre` et à traiter les relations en pipeline chaque fois que possible.
- L'étiquette (c, b) de chaque arête indique la cardinalité et le nombre de blocs pour la relation associée à cette arête.
- Une arête en trait épais indique que la relation est traitée en pipeline.

Plan n° 1 (arbre de requête R12)



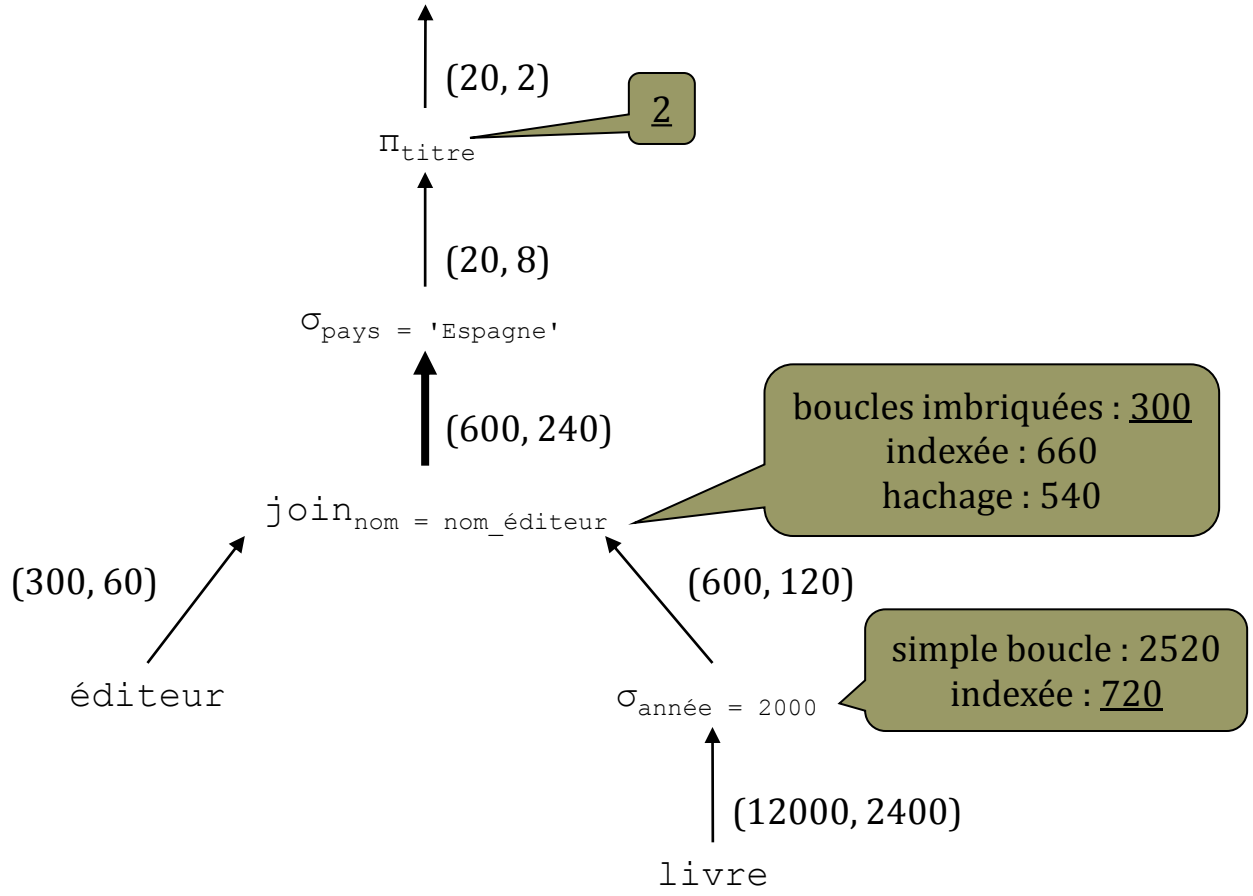
Coût total estimé = 4860 + 2 = 4862 transferts de pages

Plan n° 2 (arbre de requête R22)



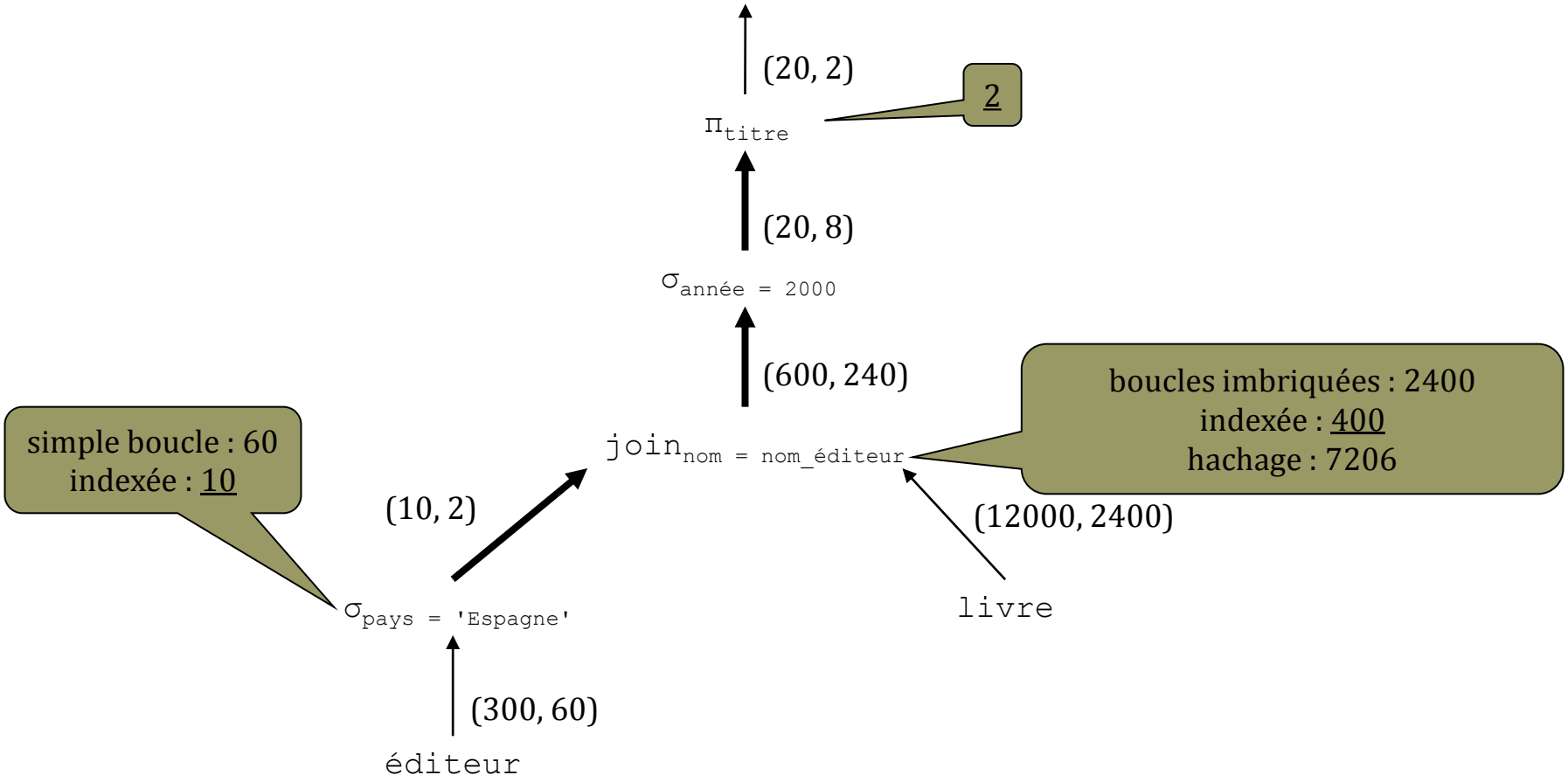
Coût total estimé = 10 + 720 + 120 + 2 = 852 transferts de pages

Plan n° 3 (arbre de requête R32)



Coût total estimé = 720 + 300 + 2 = 1022 transferts de pages

Plan n° 4 (arbre de requête R42)



Coût total estimé = 10 + 400 + 2 = 412 transferts de pages

Choix du meilleur plan

- *The winner is*: le plan n° 4 !

Concurrence et reprise

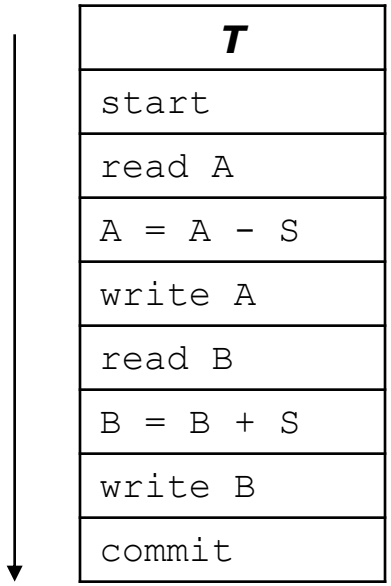
Concept de transaction

- Une **transaction** est un fragment de programme dont l'exécution fait passer une BD d'un **état cohérent** à un autre état cohérent.
- Une transaction est une suite d'**événements** dont chacun peut être :
  - la **lecture** ou l'**écriture** d'une donnée,
  - le **verrouillage** ou le **déverrouillage** d'une donnée,
  - le **démarrage** ou l'**arrêt** de la transaction.
- Une donnée est un fragment d'une BD :
  - une valeur d'attribut, un *n*-uplet, une table…

Événements

événement	**signification**
start	démarrage de la transaction
read *D*	lecture d'une donnée *D*
write *D*	modification d'une donnée *D*
rollback	annulation de la transaction
commit	confirmation de la transaction

Exemple de transaction



Propriétés d'une transaction

- **Atomicité**
  - Soit toutes les modifications effectuées par une transaction sont enregistrées dans la BD, soit aucune ne l'est.
  - Si une transaction est confirmée (`commit`) toutes les modifications qu'elle a effectuées sont enregistrées dans la BD et rendues visibles aux autres utilisateurs.
  - Si une transaction est interrompue (`rollback`) alors aucune de ces modifications n'est enregistrée dans la BD.
- **Cohérence**
  - Une transaction fait passer une BD d'un état cohérent à un autre état cohérent.
  - Un état cohérent est un état dans lequel les contraintes d'intégrité sont vérifiées.
- **Isolation**
  - Une transaction se déroule sans être perturbée par les transactions concurrentes : tout se passe comme si elle se déroulait seule.
- **Durabilité**
  - Une fois qu'une transaction a été confirmée le SGBD garantit qu'aucune modification qu'elle a effectuée ne sera perdue quelque soient les accidents qui surviendront : interruption, pannes du système d'exploitation, « crash » de disque, etc.

Problèmes dus à la concurrence

- **perte de mise à jour**
- **lecture impropre**
- **lecture non reproductible**
- **objets fantômes**

Perte de mise a jour

T_1	T_2	BD
		$A = 10$
`read A`		
	`read A`	
`A = A + 10`		
`write A`		$A = 20$
	`A = A + 50`	
	`write A`	$A = 60$

Lecture impropre (données incohérentes)

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
`read A`		
`A = A - 50`		
`write A`		$A = 70$
	`read A`	
	`read B`	
	`display A + B` (150 est affiché)	
`read B`		
`B = B + 50`		
`write B`		$B = 130$

Lecture impropre (données non confirmées)

T_1	T_2	**BD**
		$A = 50$
	`A = 70`	
	`write A`	$A = 70$
`read A` (70 est lu)		
	`rollback` (La valeur initiale de A est restaurée)	$A = 50$

Lecture non reproductible

T_1	T_2	BD
		$A = 10$
	`read A` (10 est lu)	
`A = 20`		
`write A`		$A = 20$
	`read A` (20 est lu)	

Objet fantôme

T_1	T_2	BD
		$E = \{1, 2, 3\}$
`display card(E)` 3 est affiché		
	`insert 4 into E`	$E = \{1, 2, 3, 4\}$
`display card(E)` 4 est affiché		

Exécution sérialisable

- Le principe sur lequel repose le contrôle de concurrence est celui de la **sérialisabilité** de l'exécution d'un ensemble de transactions.
- En effet, lorsque les transactions sont exécutées les unes après les autres, il n'y a pas de problèmes de concurrence.
- Appliquée systématiquement, cette solution serait très coûteuse en temps d'attente.
- La solution adoptée consiste à exécuter un ensemble de transactions concurrentes de façon à ce que le résultat soit équivalent à une exécution en série de ces transactions :
  - une telle exécution est dite **sérialisable**.

Définition de la sérialisabilité

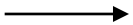
- L'exécution d'un ensemble de transactions est dite en série si, pour tout couple de transactions, tous les événements de l'une précèdent tous les événements de l'autre.
- Deux exécutions d'un même ensemble de transactions sont équivalentes si et seulement si :
  - elles sont constituées des mêmes événements,
  - elles produisent le même état final de la BD et les mêmes résultats pour les transactions :
    - les lectures produisent les mêmes résultats,
    - les écritures sont réalisées dans le même ordre.
- Une exécution concurrente d'un ensemble de transactions est dite sérialisable si et seulement si il existe une exécution en série équivalente.

Condition de sérialisabilité

- Deux opérations sont dites **conflictuelles** si elles appartiennent à deux transactions différentes et si elles ne sont pas permutables.
- Deux opérations appartenant à deux transactions différentes sont conflictuelles si et seulement si elles portent sur la même donnée et que l'une des deux au moins est une opération d'écriture.
- Une exécution concurrente est sérialisable si elle peut être transformée en une exécution en série équivalente par une suite de permutations d'opérations non conflictuelles.

Exemple d'exécution sérialisable

T_1	T_2
read A	
write A	
	read A
read B	
	write A
write B	
	read B
	write B



T_1	T_2
read A	
write A	
read B	
	read A
write B	
	write A
	read B
	write B

T_1	T_2
read A	
write A	
read B	
write B	
	read A
	write A
	read B
	write B

Verrouillage

- Le verrouillage est la technique la plus classique pour résoudre les problèmes dus à la concurrence :
  - Avant de lire ou écrire une donnée une transaction peut demander un verrou sur cette donnée pour interdire aux autres transactions d'y accéder.
  - Si ce verrou ne peut être obtenu, parce qu'une autre transaction en possède un, la transaction demandeuse est mise en attente.
- Afin de limiter les temps d'attente, on peut jouer sur :
  - la **granularité** du verrouillage : pour restreindre la taille de la donnée verrouillée,
  - le **mode de verrouillage** : pour restreindre les opérations interdites sur la donnée verrouillée.

Granularité de verrouillage

- On peut verrouiller :
  - un *n*-uplet et donc toutes ses valeurs,
  - une page et donc tous ses *n*-uplets,
  - une table et donc toutes ses lignes,
  - la BD et donc toutes ses tables.

Modes de verrouillage

- Les deux mode les plus simples sont :
  - **partagé** (S) : demandé avant de lire une donnée,
  - **exclusif** (X) : demandé avant de modifier une donnée.
- Les règles qui régissent ces deux modes sont les suivantes :
  - Un verrou partagé ne peut être obtenu sur une donnée que si les verrous déjà placés sur cette donnée sont eux même partagés :
    - commutativité lecture/lecture,
  - Un verrou exclusif ne peut être obtenu sur une donnée que si aucun verrou n'est déjà placé sur cette donnée :
    - non commutativité lecture/écriture ou écriture/écriture sur la même donnée.

Matrice de compatibilité

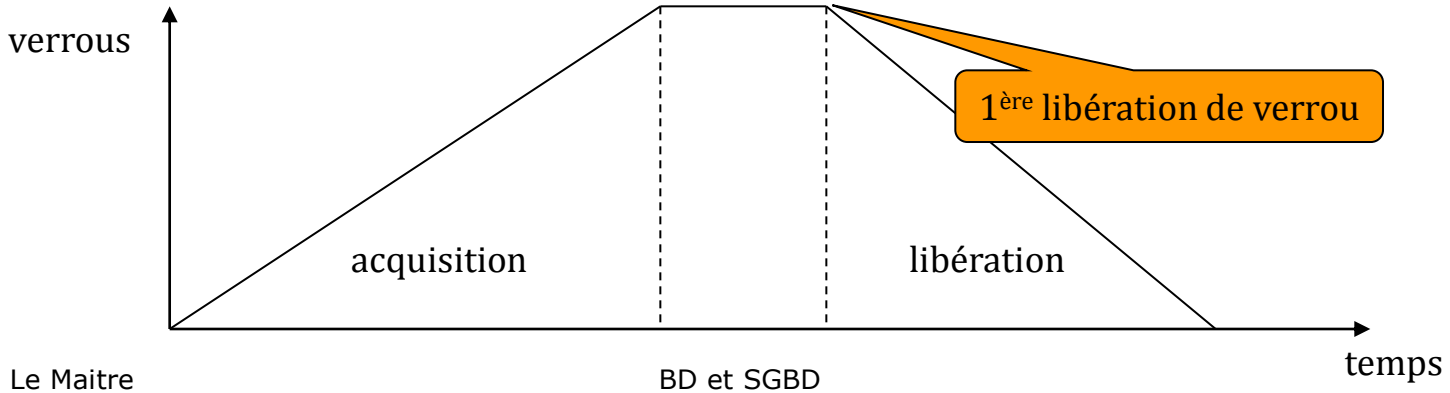
MC	***S***	***X***
S	oui	non
X	non	non

Opérations de verrouillage

Opération	**Signification**
`lock` *m D*	demande d'un verrou en mode *m* sur la donnée *D*
`unlock` *D*	déverrouillage d'une donnée *D*

Verrouillage à deux phases (1)

- Une transaction est bien formée si :
  - elle obtient un verrou sur une donnée avant de lire ou d'écrire cette donnée,
  - elle libère tous ses verrous avant de se terminer.
- Une transaction est à deux phases si elle est bien formée et si après avoir libéré un verrou elle n'en acquiert plus.
- On distingue donc :
  - une **phase d'acquisition** des verrous,
  - une **phase de libération** des verrous.



Verrouillage à deux phases (2)

- Il est démontré que l'exécution d'un ensemble de transactions à deux phases est sérialisable.
- En conséquence il ne peut y avoir ni pertes de mise à jour, ni lectures impropres, ni lectures non reproductibles.
- Les transactions sont sérialisées dans l'ordre du début de leur phase de libération.

Verrouillage à deux phases (3)

- Un verrouillage à deux phases est dit :
  - **strict**, si les verrous ne sont libérés qu'après la confirmation de la transaction,
  - **non strict** dans le cas contraire.
- L'avantage du verrouillage strict est que l'ordre de sérialisation est celui de leur point de confirmation, ce qui est intuitivement attendu par les utilisateurs.
- Dans le cas de l'abandon d'une transaction, le respect du principe d'atomicité implique que tout se passe comme si cette transaction n'avait jamais existé :
  - on impose donc que les verrous en écriture ne soient libérés qu'après la confirmation de la transaction.

Plus de perte de mise à jour

T_1	T_2	BD
		$A = 10$
`lock X A`		
`read A`		
	`lock X A`	
`A = A + 10`	*attente*	
`write A`	*attente*	$A = 20$
`unlock A`	*attente*	
	`read A`	
	`A = A + 50`	
	`write A`	$A = 70$
	`unlock A`	

Plus de lecture impropre

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
`lock X A`		
`read A`		
`A = A - 50`		$A = 70$
`write A`		
	`lock S A`	
`lock X B`	attente	
`read B`	attente	
`B = B + 50`	attente	
`write B`	attente	$B = 130$
`commit`	attente	

suite T_2

`read A`
`lock S B`
`read B`
`display A + B` (200 est affiché)

Plus de lecture non reproductible

T_1	T_2	*BD*
		$A = 10$
	`lock S A`	
	`read A` (10 est lu)	
`lock X A`		
attente	`read A` (10 est lu)	
attente	`unlock A`	
`A = 20`		
`write A`		$A = 20$

Granularité de verrouillage en SQL

- Verrouillage des tables :
  - La sérialisabilité est assurée mais l'inconvénient est la perte de concurrence due au verrouillage de tous les *n*-uplets de la table qu'ils soient ou non accédés par la requête.
- Verrouillage des *n*-uplets :
  - La concurrence est meilleure mais la sérialisabilité n'est pas assurée à cause du problème des *n*-uplets fantômes.
- Verrouillage à granularité multiple :
  - La sérialisabilité est assurée et la concurrence améliorée.

n-uplets fantômes (1)

Le verrouillage des *n*-uplets ne résoud pas le problème.

T_1	T_2

`SELECT COUNT(*)`
`FROM livre`
`WHERE année = 2003;`
(réponse *n*)	

	`INSERT INTO livre`
	`VALUES ("Les BD", 2003);`
	`COMMIT`

`SELECT COUNT(*)`
`FROM livre`
`WHERE année = 2003;`
(réponse *n* + 1)
`COMMIT`	

Chaque *n*-uplet de la table `livre` lu par T_1 est verrouillé en lecture, mais cela n'a pas d'influence sur la création d'un nouveau *n*-uplet par T_2.

n-uplets fantômes (2)

Le verrouillage des tables empêche l'apparition de *n*-uplets fantômes.

T_1	T_2

`LOCK S livre`
`SELECT COUNT(*)`
`FROM livre`
`WHERE année = 2003;`
(réponse *n*)	

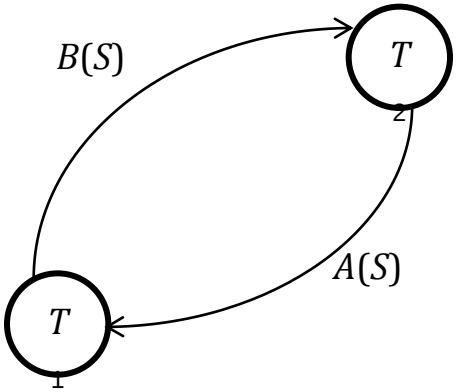
	`LOCK X livre`

`SELECT COUNT(*)`
`FROM livre`
`WHERE année = 2003;`
	attente

(réponse *n*)	*attente*
`COMMIT`	*attente*
	attente
	attente
	`INSERT INTO livre` `VALUES ("Les BD", 2003);`
	`COMMIT`

Interblocage

T_1	T_2
`lock X A`	
	`lock X B`
`lock S B`	
attente	`lock S A`
attente	*attente*



graphe d'attente

Résolution de l'interblocage

- En détectant les interblocages :
  - On inspecte à intervalles réguliers le graphe d'attente pour détecter si un interblocage s'est produit. En ce cas on défait l'une des transactions bloquées et on la relance un peu plus tard.
  - On annule une transaction dont le temps d'attente dépasse un certain seuil, et on la relance un peu plus tard.
- En évitant les interblocages :
  - On fixe un ordre sur les données et on impose aux transactions de respecter cet ordre dans leur demandes de verrous.

La norme SQL

- La norme SQL n'impose aucun choix sur l'implantation des mécanismes de gestion de la concurrence.
- Elle spécifie, entre autres :
  - la définition d'une transaction,
  - les niveaux d'isolation auxquels une transaction peut s'exécuter.

Définition d'une transaction SQL

- Une transaction est une suite de commandes SQL.
- Une transaction est démarrée par un agent lorsqu'il exécute une commande SQL et qu'il n'y a pas de transactions en cours.
- Une transaction est terminée explicitement par une commande `COMMIT` ou une commande `ROLLBACK`.
- Deux transactions ne peuvent pas être imbriquées :
  - un agent ne peut pas démarrer une transaction si sa transaction courante n'est pas terminée.

Niveaux d'isolation

- Par défaut, une transaction est exécutée en mode sérialisable.
- Cependant, ce protocole a des conséquences sur les performances d'une transaction en diminuant la concurrence.
- C'est pourquoi SQL, autorise le développeur d'une application à choisir le niveau d'isolation d'une transaction afin de maximiser la concurrence.
- Il devra bien entendu vérifier que le niveau d'isolation choisi garantit la cohérence de cette transaction.
- **Attention !** les transactions d'une même application peuvent s'exécuter à des niveaux d'isolation différents :
  - fixer le niveau d'isolation d'une transaction n'a pas d'implication sur le niveau d'isolation des autres transactions.

Les 4 niveaux d'isolation

- `READ UNCOMMITED`
  - il n'y a pas de perte de mise à jour ;
  - il peut y avoir des lectures impropres, des lectures non reproductibles, et des objets fantômes ;
  - il n'y a pas d'attente en lecture.
- `READ COMMITTED`
  - il n'y a ni perte de mise à jour, ni lecture incohérente ;
  - il peut y avoir des lectures non reproductibles et des objets fantômes ;
  - les transactions n'ont accès qu'aux données produites par des transactions confirmées.
- `REPEATABLE READ`
  - il n'y a ni perte de mise à jour, ni lecture incohérentes, ni lecture non reproductible,
  - il peut y avoir des objets fantômes.
- `SERIALIZABLE` (par défaut)
  - il n'y a ni perte de mise à jour, ni lecture incohérente, ni lecture non reproductible, ni objet fantôme,
  - l'isolation est totale, la visibilité de la BD est équivalente à un cliché réalisé au début de la transaction.

Mode d'exécution d'une transaction

- Le mode d'exécution d'une transaction est spécifié par la commande `SET TRANSACTION` qui la précède immédiatement :
  - `SET TRANSACTION`
    mode d'accès, `ISOLATION LEVEL` *niveau d'isolation*
    mode d'accès `::= READ ONLY | READ WRITE`
    niveau d'isolation `::=`
    `READ UNCOMMITED`
    `| READ COMMITTED`
    `| REPEATABLE READ`
    `| SERIALIZABLE`
- Par défaut, le niveau d'isolation est `SERIALIZABLE` et le mode d'accès est `READ ONLY`.
- **Attention !** la commande `SET TRANSACTION` n'est pas un début de transaction et elle ne peut pas être utilisée si une transaction est en cours.

Une mise en place des niveaux d'isolation (1)

- Nous montrons comment un niveau d'isolation peut être mis en place pour une transaction SQL, à l'aide de 2 types de verrous :
  - des **verrous de longue durée** qui peuvent être demandés pour la lecture ou l'écriture d'un *n*-uplet ou d'une table et qui ne sont libérés qu'après la confirmation de la transaction,
  - des **verrous de courte durée** qui sont demandés pour la lecture d'un *n*-uplet et libérés après la lecture de celui-ci.
- Soit *T* la transaction considérée.

Une mise en place des niveaux d'isolation (2)

- `READ UNCOMMITED`
  - On autorise *T* à lire un *n*-uplet sans demande préalable de verrou.
  - *T* peut donc lire une donnée verrouillée en écriture par une autre transaction et donc effectuer une lecture impropre.
- `READ COMMITTED`
  - On impose à *T* d'obtenir un verrou de courte durée sur un *n*-uplet avant de pouvoir le lire.
  - Ce verrou ne pourra pas être obtenu si une autre transaction *T'* possède un verrou en écriture sur ce *n*-uplet, et puisque *T'* ne libérera ce verrou que lors de sa confirmation (les verrous en écriture sont de longue durée), *T* ne pourra pas donc pas lire un *n*-uplet non confirmé.
  - Par contre, une autre transaction pourra modifier un *n*-uplet entre deux lectures de celui-ci par *T* : les lectures de ce *n*-uplet par *T* pourront donc ne pas être reproductibles.

Une mise en place des niveaux d'isolation (3)

- `REPEATABLE READ`
  - On impose à *T* d'obtenir un verrou de longue durée sur un *n*-uplet avant de pouvoir le lire.
  - Aucune autre transaction ne pourra donc modifier ce *n*-uplet avant que *T* soit confirmée.
  - Les lectures de ce *n*-uplet par *T* seront donc reproductibles.
- `SERIALIZABLE`
  - On impose à *T* d'obtenir un verrou en lecture de longue durée sur chaque table lue et donc sur chaque *n*-uplet de cette table.
  - Aucune autre transaction ne pourra donc insérer un *n*-uplet dans une de ces tables car cette opération est une opération d'écriture de cette table et est donc interdite si un verrou en lecture est déjà placé sur cette table, ce qui est le cas.
  - Il n'y aura donc pas de *n*-uplets fantômes dans les tables lues par *T*.

Atomicité et durabilité

- Le respect de l'atomicité peut impliquer de défaire les effets d'une transaction lorsque celle-ci a été abandonnée.
- Le respect de la durabilité implique que le SGBD doit être capable de remettre la base de données en état après une panne :
  - les mises à jour faites par une transaction non confirmée avant la panne doivent être défaites,
- C'est le gestionnaire de **reprise** qui assure cette tâche.

Reprise

- Les pannes d'un SGBD peuvent être classées en deux catégories :
  - **panne d'ordinateur**, par exemple à la suite d'une coupure de courant : les données enregistrées en mémoire centrale sont perdues mais pas celles enregistrées sur disque.
  - **panne de disque** : une partie ou toutes les données enregistrées sur ce disque sont perdues.
- La reprise, **à chaud**, après un abandon de transaction ou une panne d'ordinateur, peut être réalisée automatiquement et rapidement, en s'appuyant sur la tenue d'un journal qui garde en mémoire tous les événements d'une transaction.
- La reprise, **à froid**, après une panne de disque est plus longue à mettre en œuvre. Elle nécessite de réaliser des copies régulières de la BD et un archivage des mises à jour entre deux copies.
- Nous étudions la reprise à chaud.

Journal de transaction

- Le journal de transaction est un fichier qui contient une suite d'enregistrements dont chacun décrit un événement d'une transaction.
- Un journal est enregistré sur une **mémoire stable**, c.-à-d. une mémoire qui théoriquement ne peut pas être détruite.
- Si nécessaire, le journal est dupliqué en plusieurs exemplaires (par exemple, sur un disque miroir).
- Les enregistrements sont écrits à partir de la fin du journal (mode *append*) et donc des plus anciens aux plus récents.

Exemple d'événements journalisés

Evénement	**Signification**
(T start)	début de la transaction d'identificateur T
(T D a)	mise à jour la donnée D par la transaction T : son ancienne valeur était a
(T D n)	mise à jour la donnée D par la transaction T : sa nouvelle valeur est n
(T commit)	confirmation de la transaction T
(T abort)	abandon de la transaction T
(check point)	point de reprise

Un extrait de journal

T_1	T_2
`start`	
`read A`	
`A = A + 10`	
`write A`	
`rollback`	
	`start`
	`read B`
	`B = B + 20`
	`write B`
	`commit`

Journal
T_1 start
T_1 A 30 40
T_2 start
T_2 B 70 90
T_2 commit

Point de reprise

- Un point de reprise est une marque dans le journal qui indique qu'au moment où elle a été faite :
  - aucune transaction n'était en cours,
  - toutes les données écrites par des transactions antérieures au point de reprise avaient été transférées sur disque.
- Pour obtenir un point de reprise, il faut :
  - interdire le début de nouvelles transactions,
  - laisser se terminer les transactions en cours (c.-à-d. jusqu'à quelles aient écrit COMMIT ou ABORT dans le journal),
  - écrire sur le disque les blocs du journal encore en mémoire centrale,
  - écrire (checkpoint) dans le journal,
  - écrire sur le disque les blocs du journal encore en mémoire centrale.

Forcer l'écriture sur disque

- Une écriture dans la BD (write) ou dans le journal se fait au travers d'un tampon :
  - une donnée écrite par write l'est dans le tampon et pourra n'être écrite sur le disque que bien plus tard,
  - un enregistrement écrit dans le journal l'est dans le tampon et pourra n'être écrit sur le disque que bien plus tard.
- Pour être sûr qu'une donnée écrite par write soit enregistrée sur disque il faut **forcer l'écriture sur disque** de la page du tampon contenant cette donnée si elle ne l'a pas été.
- Pour être sûr que le journal soit entièrement enregistré sur disque, il faut **forcer l'écriture sur disque** des pages qui résident dans le tampon qui n'ont pas encore écrites sur le disque ou qui ont été modifiées.

Modes de mise à jour

- On distingue trois modes de mise à jour :
  - **Mise à jour immédiate.** Les mises à jour d'une transaction sont écrites sur disque avant la confirmation de cette transaction **.**
  - **Mise à jour différée.** Les mises à jour d'une transaction sont mémorisées le journal et ne sont écrites sur disque qu'après confirmation de cette transaction.
  - Combinaison des deux.

Mise à jour immédiate

- Les mises à jour (`write`) faites par une transaction sont écrites sur disque avant la confirmation de cette transaction.
- Si cette transaction est annulée (`rollback`) ou si une panne s'est produite avant sa confirmation, il faut annuler ces mises à jour.

Mise à jour immédiate : journalisation

- Pour chaque événement start d'une transaction T :
  - écrire (start T) dans le journal.
- Pour chaque événement write D d'une transaction T :
  - écrire (T D a) dans le journal (***write-ahead logging***) où a est l'ancienne valeur de D,
  - forcer l'écriture sur disque de D.
- Pour chaque événement commit d'une transaction d'identificateur T :
  - écrire (commit T) dans le journal.

Mise à jour immédiate : annulation d'une transaction

- Soit T la transaction à annuler.
  - dérouler le journal à l'envers jusqu'à l'enregistrement (T start),
  - dérouler le journal à l'endroit et pour chaque enregistrement ($T\ D\ a$) :
    - $D := a$
    - forcer l'écriture sur disque de D,
  - écrire (T abort)dans le journal,
  - forcer l'écriture sur disque du journal.

Mise à jour immédiate : reprise

- Dérouler le journal à l'envers jusqu'au dernier point de reprise, et pour chaque enregistrement e :
  - si e = (commit T) :
    - noter que T est confirmée,
  - si e = (abort T) :
    - noter que T est annulée,
  - si e =(T D a) et T n'est ni confirmée, ni annulée :
    - écrire l'ancienne valeur a de D dans la BD
    - noter que T est incomplète
- Pour chaque transaction T incomplète :
  - écrire (abort T) dans le journal,
  - forcer l'écriture du journal sur le disque.

Mise à jour différée

- Les nouvelles valeurs des données mises à jour par une transaction sont écrites dans le journal et ne sont écrites sur disque qu'après confirmation de cette transaction (commit).
- Une transaction non confirmée n'a donc rien écrit sur le disque :
  - Si cette transaction est annulée (rollback) ou si une panne s'est produite avant sa confirmation, il n'y a rien à faire.
  - Si une panne se produit après sa confirmation, il faut écrire les nouvelles valeurs de ces données sur disque car il n'est pas sûr qu'elles l'ont été. Ces nouvelles valeurs sont écrites dans le journal.

Mise à jour différée : journalisation

- Pour chaque événement start d'une transaction T :
  - écrire (start T) dans le journal.
- Pour chaque événement write D d'une transaction T :
  - écrire (T D n) dans le journal où n est la nouvelle valeur de D (**write-ahead logging**),
  - noter la nouvelle valeur n de D.
- Pour chaque événement commit d'une transaction T :
  - écrire (commit T) dans le journal,
  - forcer l'écriture du journal sur le disque,
  - pour chaque donnée D mise à jour par T
    - forcer l'écriture sur disque de la nouvelle valeur de D.

Mise à jour différée : reprise

- Dérouler le journal à l'envers jusqu'au dernier point de reprise et noter les transactions confirmées et les transactions incomplètes (ni confirmées, ni annulées).
- Dérouler le journal à l'endroit (depuis ce point de reprise) et pour chaque enregistrement (T D n) d'une transaction T confirmée :
  - forcer l'écriture de D sur le disque.
- Pour chaque transaction T incomplète :
  - écrire (abort T) dans le journal,
  - forcer l'écriture du journal sur le disque.