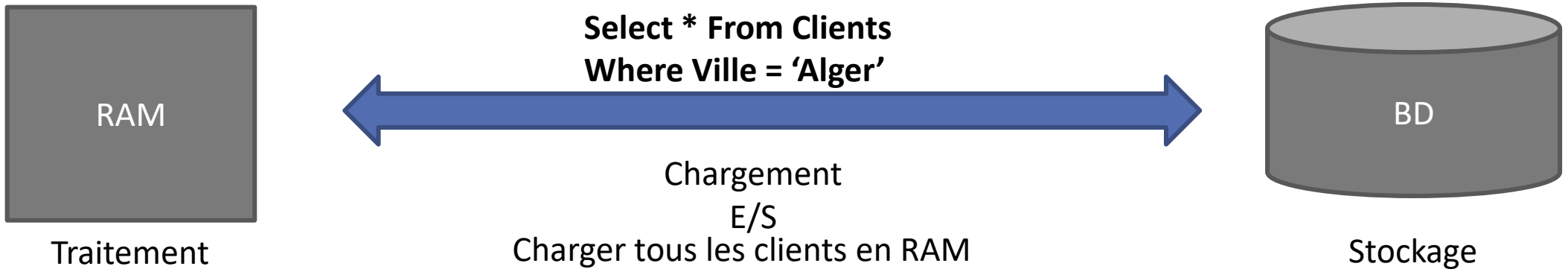
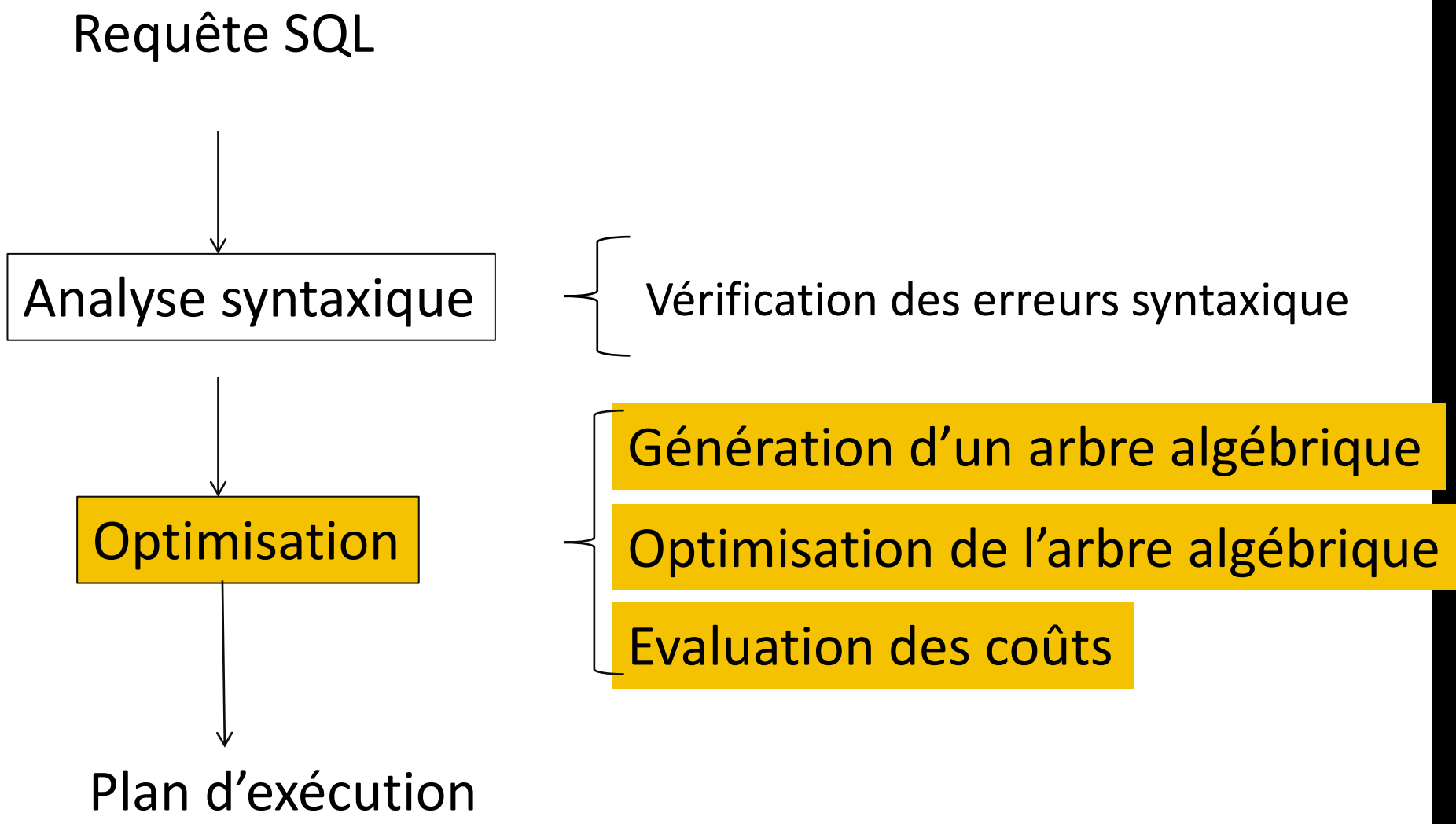
OPTIMISATION DES REQUÊTES

INTRODUCTION

- **L'exécution d'une requête nécessite:**
  - Le transfert de données de la mémoire stable MS (disques) vers la mémoire centrale MC
  - Le traitement en MC
  - Écriture si mise à jour en MS
- **Qu'est ce l'optimisation de requêtes?**
  - Définir le meilleur ordre d'exécution des opérations élémentaire d'une requête afin d'obtenir le meilleur temps d'exécution.
- **Pourquoi optimiser une requête?**
  - Requêtes complexes sur des données volumineuses implique des temps d'exécution long.



COMMENT OPTIMISER UNE REQUÊTE?



TECHNIQUES D'OPTIMISATION

- **Génération et optimisation des arbres algébriques**
  - Transformation de la requêtes en arbre algébrique pour obtenir la meilleure séquence d'opérations
  - RBO : Rules Based Optimisation
- **Evaluation des coût**
  - Coût des différentes stratégies possibles en fonction des caractéristiques des fichiers sur lesquels sont implantées les relations
  - CBO : Cost Based Optimisation

Généralement les SGBDs commerciaux combinent les deux stratégies

- Oracle utilise actuellement CBO seulement

ARBRE ALGÉBRIQUE

- **Algèbre relationnelle :** modèle formel permettant d'exprimer et de calculer les **requêtes** sur les **relations**.
- Constituée d'un ensemble **d'opérateurs algébriques :** la sélection, la jointure, la projection, les opérateurs ensemblistes.
- Le résultat de l'exécution d'un opérateur est **toujours une relation**, ce qui permet la **composition**.
- Utilisée également pour **l'optimisation de requêtes**.
- Une requête algébrique peut se présenter sous forme d'un **arbre algébrique**.

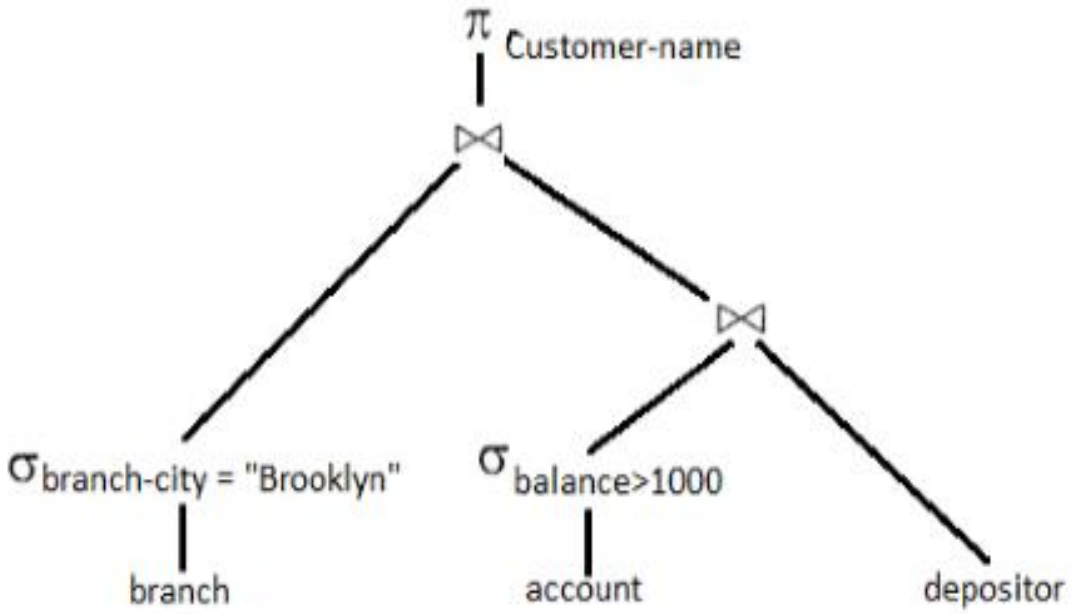
Arbre algébrique

- visualisation d'une requête sous une forme graphique d'arbre plus facile à lire qu'une forme linéaire

ARBRE ALGÉBRIQUE

Un arbre algébrique est un arbre représentant une requête

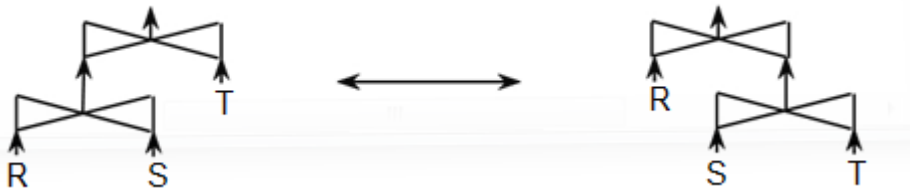
- Les **nœuds feuilles** sont les **relations de base**
- Les **nœuds intermédiaires** sont les **opérateurs**
- Le nœud **racine** est le **résultat**
- **L'arc** est un **flux** de données



PROPRIÉTÉS DES OPÉRATEURS

- Commutativité pour jointure et produit

- Associativité pour jointure et produit



- Il existe N! arbres de jointure de N relations.

PROPRIÉTÉS DES OPÉRATIONS (SUITE)

❑Groupage des projections

- ❑ $\pi A1, \ldots, An\ (\pi B1, \ldots., Bm(E)) = \pi A1, \ldots, An\ (E)$ --- {A1, …, An} inclus dans {B1, …, Bm}

❑Groupage des sélection

- ❑ $\sigma_{F1}\ (\sigma_{F2}\ (E)) = \sigma_{F1\ \wedge\ F2}(E)$

Exemple : $\sigma_{age>20}\ (\sigma_{genre='F'}$ (Étudiant)) = $\sigma_{(age>20)\ \wedge\ (genre='F')}$(Étudiant)

❑Inversion σ et π

- ❑ $\pi_{A1, \ldots, An}(\sigma_F(E)) = \sigma_F(\pi_{A1, \ldots, An}(E))$ si F porte sur A1, …, An
- ❑ $\pi_{A1, \ldots, An}(\sigma_F(E)) = \pi_{A1, \ldots, An}\ (\sigma_F(\pi_{A1, \ldots, An, B1, \ldots, Bn})\ (E))$ si F porte sur B1, …, Bm qui ne sont pas parmi A1, …, An

PROPRIÉTÉS DES OPÉRATIONS (SUITE)

Inversion Sélection-Produit

- $\sigma_F (E1 * E2) = \sigma_F (E1) * E2$ si F ne porte que sur les attributs de E1
- $\sigma_F (E1 * E2) = \sigma_{F1} (E1) * \sigma_{F2} (E2)$ Si F = F1 ∧ F2 et Fi ne porte que sur les attributs de Ei
- $\sigma_F (E1 * E2) = \sigma_{F2} (\sigma_{F1} (E1) * E2)$ si F1 porte sur les attributs de E1 et F2 sur ceux de E1 et E2
- Exemple : $\sigma_{Genre = 'F'}(\text{Étudiant} * \text{Cours}) = \sigma_{Genre='F'}(\text{Étudiant}) * \text{Cours}$

Inversion Sélection-Union

- $\sigma_F (E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$

Inversion Sélection-Diffétrence

- $\sigma_F (E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$

Inversion Projection-Produit

Inversion Projection-Union

PROPRIÉTÉS DES OPÉRATIONS (SUITE)

- **Transformation si possible du produit cartésien en jointure:**
  - Si R * S est l'argument d 'une sélection σ_F alors si F est une comparaison entre attributs de R et de S, transformer R * S en jointure : $\sigma_F (R * S) = (R \bowtie_F) S$

Attention : si F est une expression ne portant que sur les attributs de R:

- $\sigma_F (R * S) = \sigma_F (R) * S$

UN ALGORITHME D'OPTIMISATION

1. Séparer chaque sélection $\sigma_{F1 \wedge \ldots \wedge Fn}(E)$ en une cascade $\sigma_{F1}(\sigma_{F1} \ \ldots (\sigma_{Fn}(E)))$
2. Descendre chaque sélection le plus bas possible dans l'arbre algébrique
3. Descendre chaque projection aussi bas possible dans l'arbre algébrique
4. Combiner des cascades de sélection et de projection dans une sélection seule, une projection seule, ou une sélection suivie par une projection
5. Regrouper les nœuds intérieurs de l'arbre autour des opérateurs binaires ($\bowtie$, *, -, $\cup$). Chaque opérateur binaire crée un groupe
6. Produire un programme comportant une étape pour chaque groupe, à évaluer dans n'importe quel ordre mais tant qu'aucun groupe ne soit évalué avant ses groupes descendants.

QUELQUES PROBLÈMES

- La restructuration d'un arbre sous-entend souvent la permutation des opérateurs binaires
  - Cette optimisation ignore la taille de chaque table, les facteurs de sélectivité, etc.
  - Aucune estimation de résultats intermédiaires
- Exemple:

(Ouvrier ⋈ Usine) ⋈ Contrat

Si cardinalité (Usine) <<<Cardinalité(Contrat) << Cardinalité(Ouvrier) alors l'expression suivante a des chances d'être plus performante:

(Usine ⋈ Contrat) ⋈ Ouvrier

→ Optimisation basée sur les modèles de coût

MÉTHODES D'OPTIMISATION BASÉES SUR LES MODÈLES DE COÛTS

NOTIONS UTILES

- Les tables relationnelles sont stockées physiquement dans des fichiers sur disque

- Lorsqu'on veut accéder aux données, on doit transférer le contenu pertinent des fichiers dans la mémoire

  - Fichier = séquence de tuples
  - Bloc (page) = unité de transfert de données entre disque et mémoire
  - Facteur de blocage = nombre de tuples d'une relation qui peuvent contenus dans un bloc
  - Coût de lecture (ou écriture) d'une relation = nombre de page à lire du disque vers la mémoire (ou à écrire de la mémoire vers le disque)

ESTIMATION DU COÛT DES OPÉRATIONS PHYSIQUES

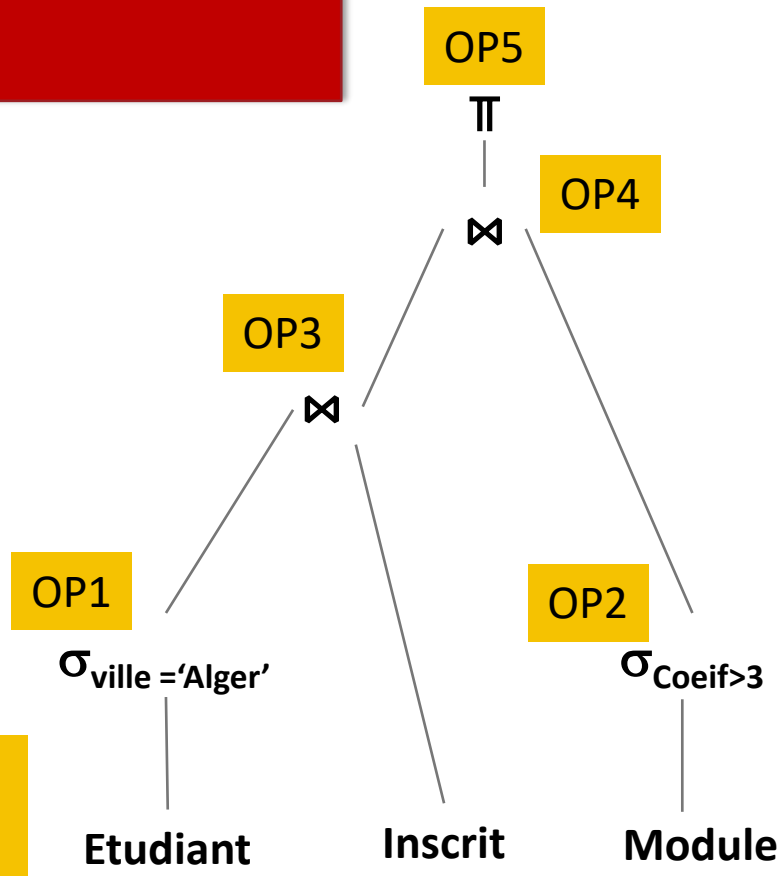
- **TempsES** : Temps d'accès à mémoire secondaire (MS)
- **TempsUC:** Souvent négligeable
- **TailleMC** : espace mémoire centrale
- **TailleMS** : espace mémoire secondaire

STATISTIQUES AU SUJET DES TABLES

Statistique	Signification
||T||	Nombre de lignes de la table T
T_Tuple	La taille d'une ligne de la table T
T_page	La taille d'une page en octets
|T|	Nombre de pages de la table T
FBT	Facteur de blocage moyen de T
Val (T,a)	Nombre de valeurs distinctes de la colonne a pour la table T
MinT(colonne)	Valeur minimum de la colonne de T
MaxT(colonne)	Valeur maximum de la colonne de T

COÛT D'UNE REQUÊTE

- Soit R une requête composée de n opérations algébriques (Opi)
- Coût (R) = ∑ Coût (OPi)
- Coût (Opi) = Nombre de pages transférées de la MS vers la RAM
- Le Coût (Opi) dépend de la taille des entrées de Opi et de l'algorithme de Opi (avec ou sans index, hachage, etc)



- Taille d'un tuple T_Tuple (Opi):
  - *T_Tuple (σ(T))= T_Tuple (T)*
  - *T_Tuple (T1 ⋈T2)=T_Tuple(T1)+T_tuple(T2)*
  - *T_Tuple ($\Pi_{A1,A2,...}(T)$) =∑Taille (Ai)*
- Nombre de tuple ||Opi||:
  - *||σ(T)||= FS * ||T||*
  - *||T1 ⋈T2||= FS * ||T1||* ||T2||*
  - *||$\Pi_{A1,A2,...}(T)$)||=||T||*

FS est le facteur de Sélectivité?

FACTEUR DE SÉLECTIVITÉ (FS)

- **Désigne le pourcentage de lignes sélectionnées sur le nombre total de tuples**
  - Entre 0 et 1
  - Opération trop sélective : **FS tend vers zéro**
  - Opération non sélective : **FS tend vers 1**
- **Exemple :**
  - Si étudiants habitant Alger représentent **20 %**
  - Le facteur de sélectivité du prédicat **Ville='Alger'** est de **0,2**
  - Si la table Etudiant contient **100 000 étudiants**, la requête

    Select * from Etudiant where Ville = Alger

    retournera **20 000 tuples**

FACTEUR DE SÉLECTIVITÉ POUR LA RESTRICTION

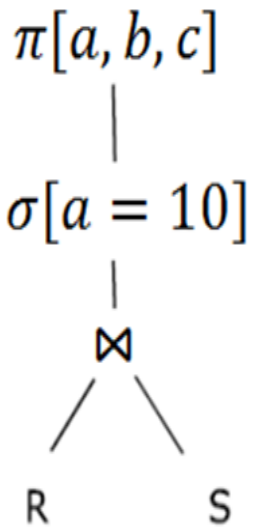
- **||(σ(R))|| = SF * ||R||**
  - FS (A = valeur) = 1 / Val(A)
  - FS(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
  - FS(A < valeur) = (valeur - min(A)) / (max(A) - min(A))
  - FS(A IN liste valeurs) = (1/Val(A)) * CARD(liste valeurs)
  - FS(P et Q) = FS(P) * FS(Q)
  - FS(P ou Q) = FS(P) + FS(Q) - FS(P) * FS(Q)
  - FS(not P) = 1 - FS(P)

FACTEUR DE SÉLECTIVITÉ POUR LA JOINTURE

- La taille d'une jointure est estimée par la formule suivante:
  - **$||R1 \bowtie R2|| = FS * ||R1|| * ||R2||$**
- Produit cartésien : **FS=1**
- Une estimation de FS si jointure quelconque
  - **$FS\ (R1\ {}_a\bowtie_b\ R2) = 1/Max(Val(a,R1), Val(b,R2))$**
- Jointure sur clé étrangère sur R1 : **$||R1 \infty R2|| = ||R1||$**
- **Le coût d'exécution de la jointure dépend de l'algorithme de jointure.**

EXERCICE

- **Soit un arbre algébrique et des informations statistiques sur les tables qu'il manipule.**



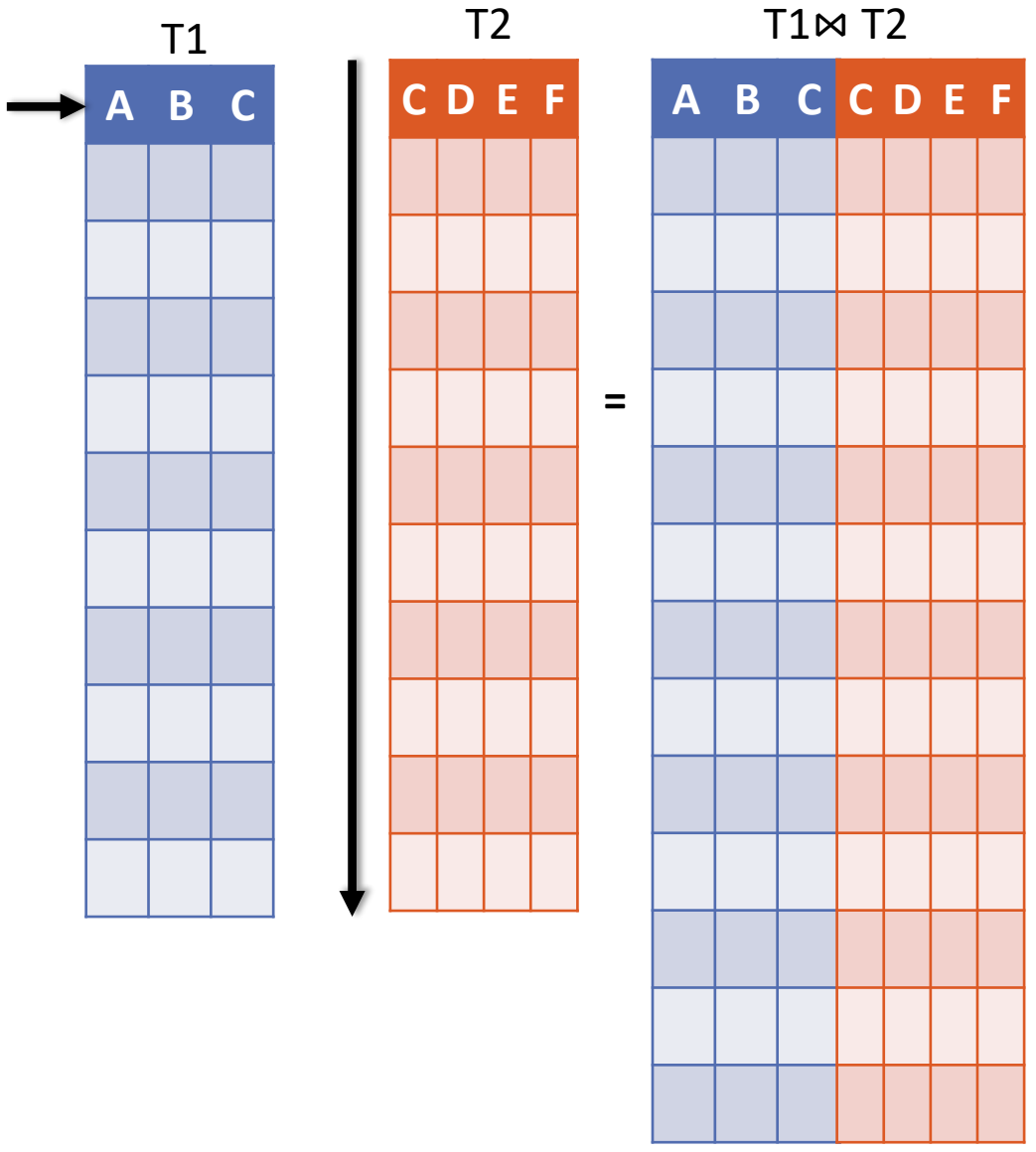
Statistique sur R:
||R||=5000
Val(R,a)= 50
Val(R,b)=100

Statistique sur S:
||S||=2000
Val(S,b)= 200
Val(S,c)=100

- **Indiquez les exécutions possibles et évaluez la cardinalité du résultat après chaque opération.**

ALGORITHMES DE JOINTURE

- Jointure sans index
  - Jointure par boucles imbriquées (nested loop)
  - Jointure par tri-fusion (sort-join)
  - Jointure par hachage (hash-join)
- Jointure avec index
  - Jointure avec boucles indexées



JOINTURE PAR BOUCLES IMBRIQUÉES (JBI)

❑ **Principe: R ⋈ c S**

- ❑ La première table est lue séquentiellement et de préférence stockée entièrement en mémoire
- ❑ Pour chaque tuple de R, il y a une comparaison avec les tuples de S

```
POUR chaque ligne lR de R
        POUR chaque ligne lS de S
                SI lR.c = lS.c
                        Produire la ligne concaténée à partir de lR et lS
                FINSI
        FINPOUR
FINPOUR
```

Coût d'exécution: |R|+|R|*|S| opérations de lecture (|R|: nombre de page de R)

Cardinalité du résultat : ||R ∞ S|| = ||R|| * ||S||/max {Val(R, c), Val(S, c)};

Remarque: La cardinalité du résultat est toujours la même quelque soit le type de jointure, c'est le coût d'exécution qui diffère.

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Source de l'exemple: Cours de Mme Azzouz

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6
F3	V2	C4
F3	V2	C5

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6
F3	V2	C4
F3	V2	C5
F4	V3	C1

Jointure par boucles imbriquées

Exemple : Fournisseur $\underset{\text{ville}}{\bowtie}$ **Client**

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6
F3	V2	C4
F3	V2	C5
F4	V3	C1
F5	V3	C1

JOINTURE PAR TRI FUSION

- **Principe**
  - **Trier les deux tables sur les colonnes de jointure**
  - **Effectuer la fusion**
  - **Complexité: Le tri qui coûte cher**
- **Plus efficace que les boucles imbriquées pour les grosses tables**

Coût =

- **|T1|+|T2|** si T1, T2 déjà triées
- **$2*|T1|*\log_b(|T1|)+|T2|+|T1|$** Si |T2| triée
- **$2*|T2|*\log_b(|T2|)+|T2|+|T1|$** Si |T1| triée
- **$2*|T1|*\log_b(|T1|)+2*|T2|*\log_b(|T2|)+|T2|+|T1|$** si aucune table n'est triée
- **b** : nombre de pages du tampon en mémoire centrale disponible pour le tri

JOINTURE PAR TRI FUSION - ALGORITHME

Trier R et S et réécrire dans des fichiers temporaires

Lire groupe de lignes GR(cR) de R pour la première valeur cR de clé de jointure

Lire groupe de lignes GS(cS) de S pour la première valeur cS de clé de jointure

TANT QUE il reste des lignes de R et S à traiter

SI cR = cS

Produire les lignes concaténées pour chacune des combinaisons de lignes de GR(cR) et GS(cS);

Lire les groupes suivants GR(cR) de R et GS(cS) de S;

SINON

SI cR < cS

Lire le groupe suivant GR(cR) de R

SINON

SI cR > cS

Lire le groupe GS(cS) suivant dans S

FINSI

FINSI

FINSI

FIN TANT QUE

Jointure par tri fusion

Exemple : Fournisseur $\bowtie_{ville}$ Client

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

4 groupes (V1,V2,V3,V4)

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

3 groupes (V1,V2,V3)

F1, V1 ⋈ C2,V1 / C3,V1 / C6,V1 → F1 V1 C2, F1 V1 C3, F1 V1 C6, F2 V1 C2, F2 V1 C3, F2 V1 C6

F3, V2 ⋈ C4,V2 / C5,V2 → F3 V2 C4, F3 V2 C5

F4,V3 / F5,V3 ⋈ C1, V3 → F4 V3 C1, F5 V3 C1

F6, V4

On s'arrête

Nombre d'accès : 6(charger Fournisseur)+6(charger client)= 12 lectures

Avantage : on ne revient jamais en arrière à cause du tri.

Si boucle imbriquée: Nombre max d'accès = 6+6*6=42 lectures

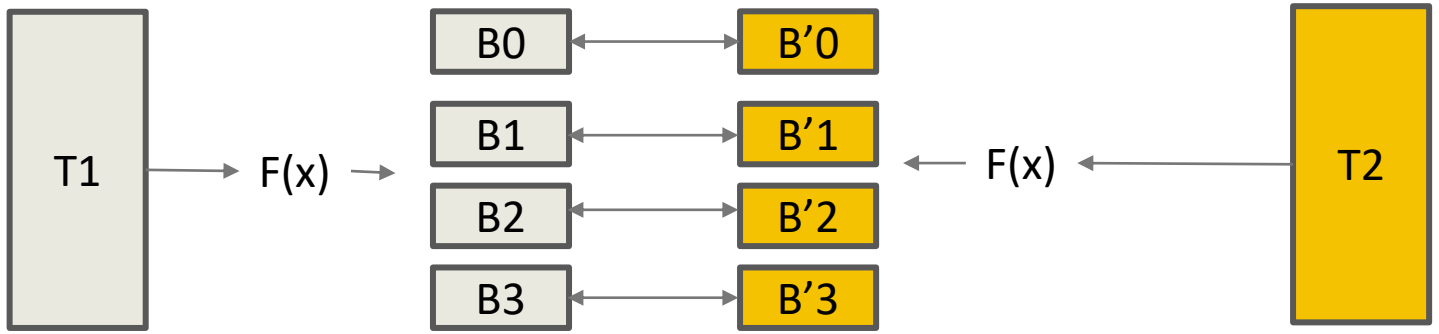
Source de l'exemple: Cours de Mme Azzouze

PRINCIPE DE LA JOINTURE PAR HACHAGE

1. **Hacher la plus petite des deux tables en n fragments**

2. **Hacher la seconde table, avec la même fonction, en n autres fragments**

3. **Réunir fragments par paire, et faire la jointure entre les fragments**



Exemple : fonction de hachage Modulo 4

COÛT JOINTURE PAR HACHAGE

1. **Chargement de T1 : |T1|**
2. **Hachage de T1, création des groupes en RAM : 0**
3. **Déchargement des groupes en MS : |T1|**
4. **Chargement de T2 : |T2|**
5. **Hachage de T2, création des groupes en RAM : 0**
6. **Déchargement des groupes en MS : |T2|**
7. **Chargement du Block Bi de T1 avec B'i de T2, faire la jointure entre les deux blocs : |Bi|+|B'i|**

Au total : $\sum|Bi|+|B'i|=|T1|+|T2|$

Coût Total $= 3(|T1|+[T2|)$

Exemple : Fournisseur $\bowtie_{ville}$ Client
Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Source de l'exemple: Cours de Mme Azzouze

Exemple : Fournisseur $\bowtie_{ville}$ Client

Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Partitions de la table fournisseur

1	2	3	4	0
F1 V1 F2 V1	F3 V2	F4 V3 F5 V3	F6 V4	

Exemple : Fournisseur $\bowtie_{ville}$ Client

Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Partition de la table fournisseur

1	2	3	4	0
F1 V1 F2 V1	F3 V2	F4 V3 F5 V3	F6 V4	

Partitions de la table Client

1	2	3	4	0
C2 V1 C3 V1 C6 V1	C4 V2 C5 V2	C1 V3		

Exemple : Fournisseur $\bowtie_{ville}$ Client

Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Partition de la table fournisseur

1	2	3	4	0
F1 V1 F2 V1	F3 V2	F4 V3 F5 V3	F6 V4	

Partition de la table Client

1	2	3	4	0
C2 V1 C3 V1 C6 V1	C4 V2 C5 V2	C1 V3		

Exemple : Fournisseur $\bowtie_{ville}$ Client

Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6
F3	V2	C4
F3	V2	C5

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Partition de la table fournisseur

1	2	3	4	0
F1 V1 F2 V1	F3 V2	F4 V3 F5 V3	F6 V4	

Partition de la table Client

1	2	3	4	0
C2 V1 C3 V1 C6 V1	C4 V2 C5 V2	C1 V3		

Exemple : Fournisseur $\bowtie_{ville}$ Client

Fonction d'hachage h(Ville)=Ville[1] mod 5

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3
F6	V4

Numf	Ville	Numc
F1	V1	C2
F1	V1	C3
F1	V1	C6
F2	V1	C2
F2	V1	C3
F2	V1	C6
F3	V2	C4
F3	V2	C5
F4	V3	C1
F5	V3	C1

NumC	VilleC
C2	V1
C3	V1
C6	V1
C4	V2
C5	V2
C1	V3

Partition de la table fournisseur Partition de la table Client

1	2	3	4	0
F1 V1 F2 V1	F3 V2	F4 V3 F5 V3	F6 V4	

1	2	3	4	0
C2 V1 C3 V1 C6 V1	C4 V2 C5 V2	C1 V3		

BOUCLE INDEXÉE

$T1 \ {}_a\bowtie_b \ T2$

- Comme boucles imbriquées mais avec l'utilisation de l'index défini sur l'attribut b de T2

1. Charger une partie de T1 en RAM
2. Charger l'index
3. Pour chaque tuple de T1, parcourir l'index, retourner les RID des tuples de T2 à joindre
4. Charger ces tuples en RAM
5. Concaténer les tuples de T1 et T2

Coût = **|Index|+|T1|+||T1||*|T2|/Val(T2,b)**

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

Source de l'exemple: Cours de Mme Azzouze

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

NBR Accès

Lire Fournisseur (1) et accéder à C2, C3, C6 → **1+3**

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

NBR Accès

Lire Fournisseur (1) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (2) et accéder à C2, C3, C6 → **1+3**

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

NBR Accès

Lire Fournisseur (1) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (2) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (3) et accéder à C4, C5 → **1+2**

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

NBR Accès

Lire Fournisseur (1) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (2) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (3) et accéder à C4, C5 → **1+2**

Lire Fournisseur (4) et accéder à C1 → **1+1**

Jointure avec Index : Recherche indexée

NumF	VilleF
F1	V1
F2	V1
F3	V2
F4	V3
F5	V3

NumC	VilleC
C1	V3
C2	V1
C3	V1
C4	V2
C5	V2
C6	V1

Index sur l'attribut ville de la table client:
A chaque lecture d'un tuple de Fournisseur accéder directement au tuple correspondant dans la table Client

Nombre d'accès : 5 + 5*6/ 3 = 15

3 = nombre d'entrées de l'index ou nombre de valeur distinct d e ville dans Client

NBR Accès

Lire Fournisseur (1) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (2) et accéder à C2, C3, C6 → **1+3**

Lire Fournisseur (3) et accéder à C4, C5 → **1+2**

Lire Fournisseur (4) et accéder à C1 → **1+1**

Lire Fournisseur (5) et accéder à C1 → **1+1**

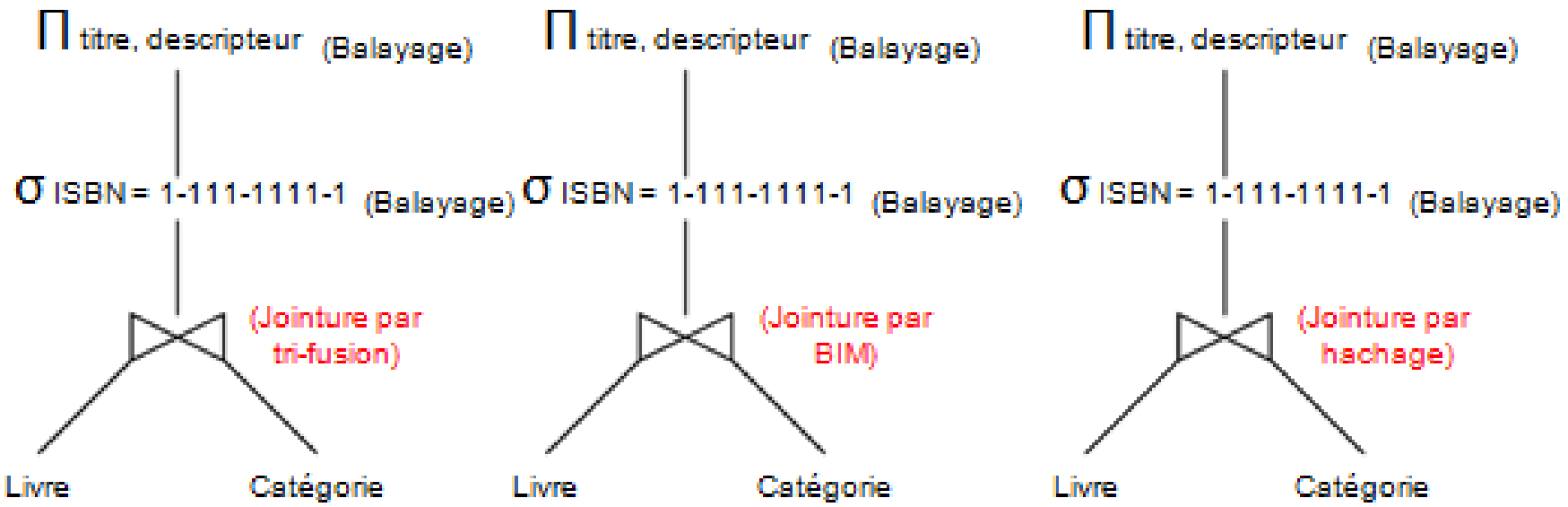
Conclusion sur les algorithmes de jointure

- Si une des deux relations rentre en mémoire c'est les boucles imbriquées qui est plus efficace (accès à une seule relation).
- Dans le cas de grandes relations (non stockées en mémoire centrale) l'algorithme basé sur le tri-fusion est plus efficace que les boucles imbriquées.
- L'efficacité de l'utilisation de l'index dépend de la sélectivité de celui-ci, c'est-à-dire de **son nombre d'entrées** (nombre de valeurs distinctes de l'argument de l'index).
- L'efficacité de la jointure avec hachage par rapport aux boucles imbriquées dépend du nombre de fragments résultat du hachage.

Source : Cours de Mme Azzouze

PLUSIEURS PLANS D'EXÉCUTION POUR UN ARBRE ALGÉBRIQUE

- **Pour chaque opération logique**
  - plusieurs choix d'opérations physiques



OPTIMISATION BASÉE SUR MODÈLE DE COÛT

- **Soit Q une requête à optimiser**
- **Procédure**

1. **Énumérer tous les plans {P_1, …, P_m} pour chaque requête (notons que chaque requête possède un ensemble d'opérations O_1, …, O_k)**
2. **Pour chaque plan P_j**
   - Pour chaque opération O_i du plan P_j, énumérer les opérations physiques possibles
   - Sélectionner l'opération physique ayant le coût le moins élevé .
   - **Coût(P_i) = $\Sigma_{(l=1, k)}$ min(O_l)**
3. **Le meilleur plan est le plan avec moindre coût.**

CHOIX DU MEILLEUR PLAN

- La recherche exhaustive du meilleur plan d 'exécution -> opération couteuse
- Solution approchée -> heuristiques
- Choisir la meilleure estimation approximative du coût

Devoir:

- **Comment utiliser les heuristiques pour choisir le meilleur plan d'exécution d'une requêtes SQL.**
- **Etudier un exemple d'heuristique (ex: recuit simulé, algorithme génétique, etc.) appliqué à ce problème.**

PLAN D'EXÉCUTION SOUS ORACLE

- Pour récupérer le plan d'exécution d'une requête sous SQLplus:

```
> Explain plan for  Select …. ;
```

- Le plan est stockée dans la table PLAN_TABLE

- Pour afficher le plan généré:

```
>  Select * from table(dbms_xplan, display);
```

PLAN D'EXÉCUTION SOUS ORACLE

```
SQL> explain plan for SELECT FILM.TITRE
  2                       FROM FILM, VU
  3                      WHERE FILM.TITRE=VU.TITRE;

Explicit.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1238637468

-----------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT |       |     8 |    80 |     1   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN | PK_VU |     8 |    80 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------

8 lignes slectionnes.
```

Question?