Université Abou Bakr Belkaid Tlemcen
Faculté des Sciences
Département d'informatique

# Administration des Bases de Données

L3 2017-2018

Mr H.MATALLAH

# Administration des Bases de Données

1. Notions fondamentales
2. SQL Avancé
3. Gestion d'intégrité et de cohérence
4. Vues et Index
5. Optimisation des requêtes
6. Gestion des transactions : Gestion des accès concurrents

# CHAPITRE 5

# OPTIMISATION DES REQUÊTES

# Plan Chapitre 5

1. Définition
2. Approches pour optimiser
3. Arbre algébrique
4. Exemple fixant la problématique
5. Traitement d'une requête
6. Stratégie générale d'optimisation
7. Optimisation basée sur les modèles de coût

# Optimisation des Requêtes

## *Définition*

- **Optimisation :** Amélioration, maximalisation, rationalisation
- **Optimisation :** Donner le meilleur rendement possible
- **Optimisation en informatique :**
  - L'optimisation de code permet d'améliorer les performances d'un logiciel
  - L'optimisation pour les moteurs de recherche permet d'améliorer le classement d'un site web dans les résultats d'une requête sur les moteurs de recherche
  - Dans les bases de données, l'optimisation des bases de données et en particulier **l'optimisation de requête** définit la meilleure exécution par le SGBD d'un code donné

# Optimisation des Requêtes

## *Définition*

- Trop souvent, on ne se soucie d'optimisation que lorsque :
  - L'application ne fonctionne pas
  - Les utilisateurs sont mécontents
  - La situation est explosive !
- Pourtant plus de 90% de l'optimisation possible provient de l'application et du design de la base de données
- Malheureusement, il n'existe pas le paramètre **PERFORMANCE=MAXI** dans les paramètres d'initialisation d'une base

# Optimisation des Requêtes

## *Approches pour optimiser*

- **Optimisation préventive** : Lors de la conception de la base
  - Optimisation du modèle conceptuel
- **Optimisation curative** : Lors de l'exploitation de la base
  - Optimisation des requêtes

# Optimisation des Requêtes

- ***Optimisation préventive : Optimisation du modèle conceptuel***
  - **Choix du modèle**
  - **Normalisation**
  - **Dénormalisation**

# Optimisation des Requêtes

## *Optimisation préventive : Optimisation du modèle conceptuel*

- **Choix du modèle**
  - Choix de l'architecture (modèle relationnel) : permet d'éviter la duplication d'information et donc d'avoir des données plus consistantes
  - Choix des types des champs : Variables plus adaptées au besoin (Char ou Num, Entier ou Décimale à virgule fixe ou Flottante, Date ou Texte)
  - Utilisation de champs à longueur variable au besoin (Ex: Varchar au lieu de Char)
  - Faire un bon indexage des tables

# Optimisation des Requêtes

## *Optimisation préventive : Optimisation du modèle conceptuel*

- **Normalisation**
  - Fragmentation des données dans plusieurs tables
  - Consiste au respect de certaines contraintes appelées formes normales, qui s'appuient sur les dépendances fonctionnelles entre attributs
  - Un des grands principes est que chaque donnée ne doit être présente qu'une seule fois dans la base, sinon un risque d'incohérence entre les instances de la donnée apparaît
  - **Objectif** : Avoir un "bon" modèle de données dans lequel les données seront bien organisées et consistantes, faciliter la mémorisation des données en minimisant la redondance des données et les problèmes sous-jacents de mise à jour ou de cohérence

# Optimisation des Requêtes

## *Optimisation préventive : Optimisation du modèle conceptuel*

- **Dénormalisation**
  - Simplification du schéma en regroupant plusieurs tables liées par des références, en une seule table, en réalisant statiquement les opérations de jointure adéquates
  - Consiste à la duplication délibérée de certaines données
  - Elle s'avère utile si : les requêtes les plus importantes portent sur des données réparties sur plusieurs tables, des calculs doivent être effectués sur une ou plusieurs colonnes avant que la requête ne renvoie une réponse, les tables doivent être consultées de différentes manières par différents utilisateurs simultanément, certaines tables sont très fréquemment utilisées
  - Un schéma doit être dénormalisé lorsque les performances de certaines recherches faibles et que cette dégradation à pour cause des jointures
  - **Objectif** : Améliorer les performances de la BD en accélérant l'extraction des données, en implémentant les jointures plutôt qu'en les calculant

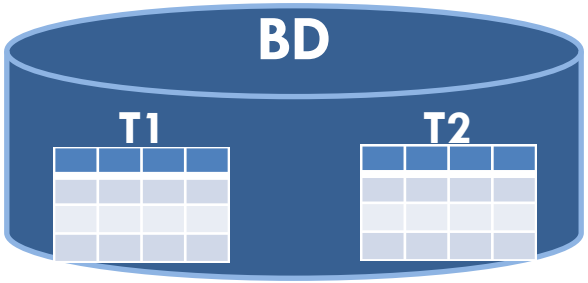
# Optimisation des Requêtes

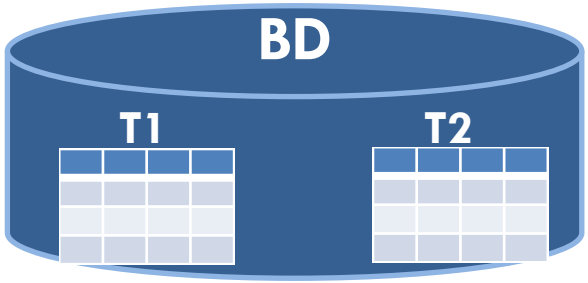
## *Optimisation curative : Optimisation des requêtes*

- Parmi les objectifs des BD est de restituer l'information dans des **délais acceptables**
- **Optimisation** : Trouver la meilleure façon d'accéder et de traiter les données afin de répondre à la requête
- La plupart des SGBD adoptent des langages non procéduraux qui ne fournissent pas les chemins d'accès aux données, ainsi l'optimisation devient l'affaire du système
- Tout SGBD implante un ensemble de techniques de base d'optimisation et dispose d'un module d'évaluation de requêtes qui permet de choisir la meilleure stratégie possible d'exécution d'une expression relationnelle

# Optimisation des Requêtes

- ***Optimisation des requêtes***




Q ↑ Tuples ↓

**Requête Q :**

```
Select T1.A, T2.B
From T1,T2
Where T1.C=T2.C
```

Q ↑ Tuples ↓

**Exécutée en 10ms !!** — **USER1**

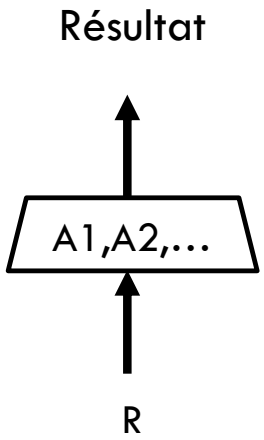
**Une même requête peut s'exécuter de différentes manières**

**Exécutée en 110ms !!** — **USER2**

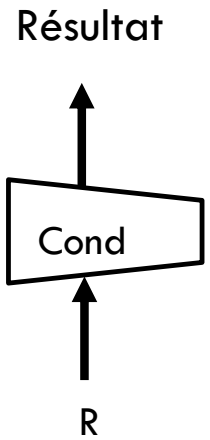
# Optimisation des Requêtes

- ***Représentation graphique des opérateurs***

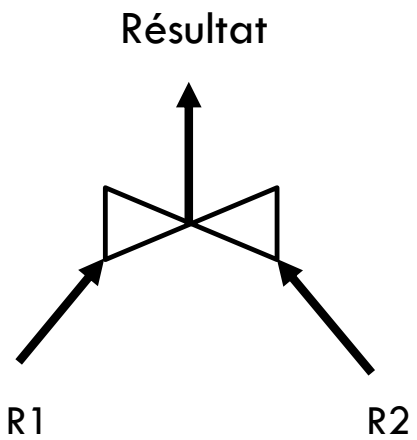
- **Projection**



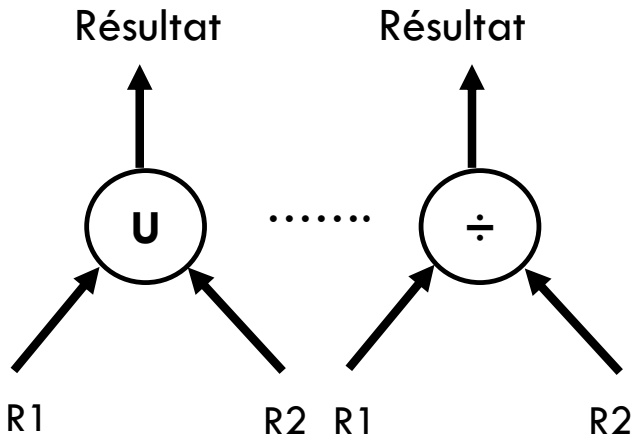

- **Sélection**




- **Jointure**




- **5 Autres opérateurs**



# Optimisation des Requêtes

## *Arbre algébrique*

- L'arbre algébrique ou l'arbre de requête est une description graphique plus lisible à traduire

- Un arbre de requête est une structure de données arborescente qui correspond à une expression de l'algèbre relationnelle :
  - Les relations fournies en entrée à la requête sont présentées sous forme de nœuds feuilles dans l'arbre
  - Les opérateurs de l'algèbre relationnel sont présentées sous forme de nœuds internes

- L'arbre algébrique illustre **l'ordre d'exécution des opérateurs**

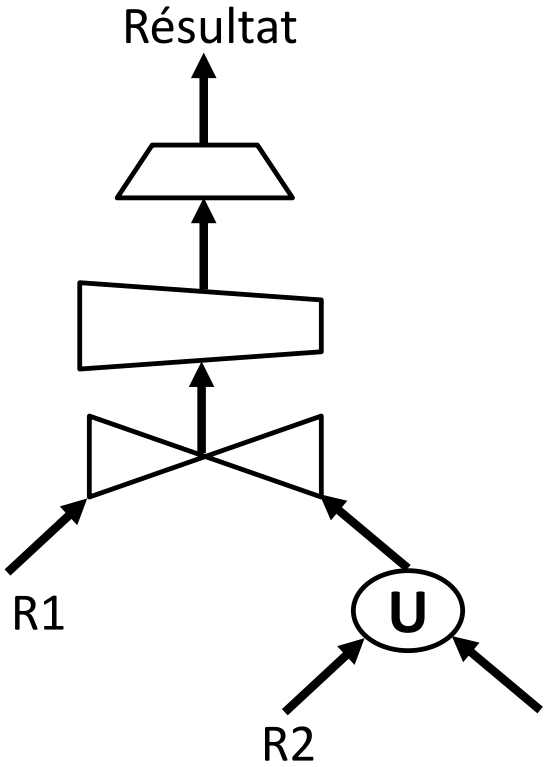
R3

# Optimisation des Requêtes

## *Arbre algébrique*

- **Exemple :**

1. Union
2. Jointure
3. Sélection
4. Projection



# Optimisation des Requêtes

## *Exemple fixant la problématique*

- Soit la base de données du TD avec les hypothèses suivantes :
  - Cardinalité Fournisseur = 100
  - Cardinalité Commande = 10 000

    *Cardinalité d'une table : Nombre de lignes*
    *Degré d'une table: Nombre de colonnes*

- Soit la requête : « ***Donner les noms des fournisseurs qui ont fournit le produit P2999*** »
  - Requête SQL :

```
SELECT Nom
FROM Fournisseur, Commande
WHERE PNum = 'P2999' AND Fournisseur. FNum = Commande. FNum
```

# Optimisation des Requêtes

## *Exemple fixant la problématique*

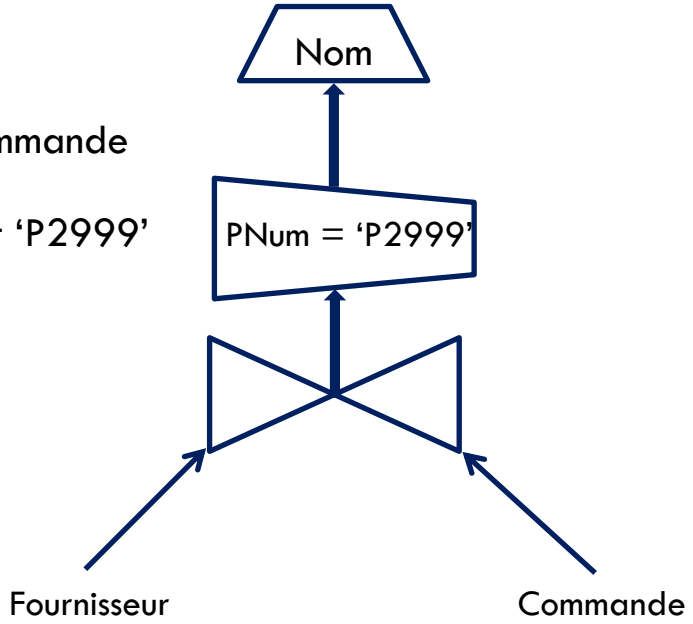
- Un arbre algébrique non optimisé de la requête :

**Ordre d'exécution des opérateurs**

$T_1 \leftarrow$ Joindre la relation Fournisseur avec Commande

$T_2 \leftarrow$ Sélectionner de $T_1$ les tuples du produit 'P2999'

$T_3 \leftarrow$ Projeter $T_2$ sur Nom



# Optimisation des Requêtes

## *Exemple fixant la problématique*

- **Evaluation de l'arbre non optimisé :**
  - 1ére étape : Jointure des relations Fournisseur et Commande
    - lecture des 10 000 tuples de commandes,
    - puis lecture de chacun des 100 fournisseurs 10 000 fois,
    - construction du résultat intermédiaire de jointure (10 000 tuples joints)

    *(On n'est pas sûr que tous les résultats peuvent être gardés en mémoire)*
  - 2éme étape : Restriction du résultat de l'étape précédente
    - les seuls tuples vérifiant PNum='P2999' seront gardés. Nous supposons qu'ils sont 50 et peuvent être gardés en mémoire
  - 3éme étape : projection sur le Nom du Fournisseur
    - parcours des 50 tuples du résultat précédent afin de ne garder que Nom
- Perte de temps en effectuant la jointure en premier du fait des lectures répétées

# Optimisation des Requêtes

## *Exemple fixant la problématique*

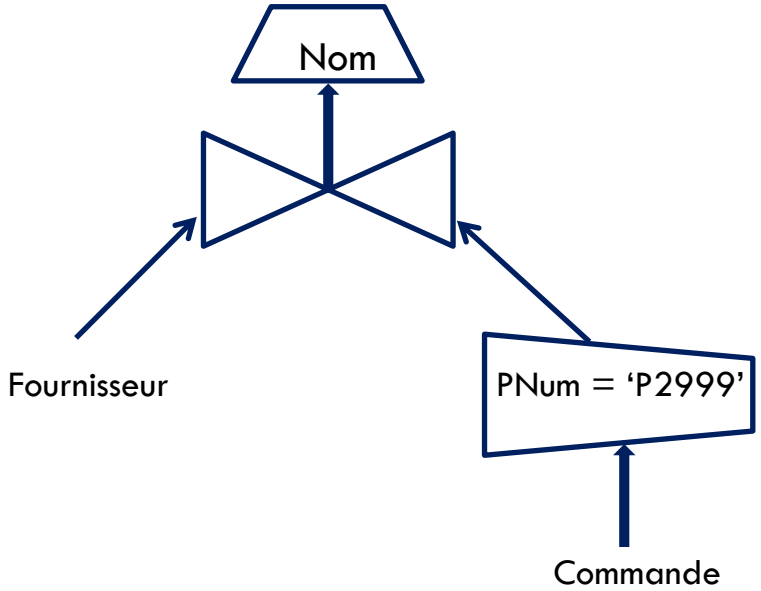
- Un arbre algébrique optimisé de la requête :

**Ordre d'exécution des opérateurs**

$T_1 \leftarrow$ Sélectionner les Commande du produit 'P2999'

$T_2 \leftarrow$ Joindre $T_1$ avec la relation Fournisseur

$T_3 \leftarrow$ Projeter $T_2$ sur Nom



# Optimisation des Requêtes

## *Exemple fixant la problématique*

- **La deuxième stratégie est beaucoup plus efficace :**

  - 1ére étape : Effectuer la restriction de Commande à PNum= 'P2999'

    - lecture des 10 000 tuples de commande pour ne garder en mémoire qu'une table de 50 tuples

  - 2éme étape : Effectuer la jointure du résultat précédent avec Fournisseur

    - cette étape entraîne l'extraction des 100 fournisseurs . Le résultat contient 50 tuples

  - 3éme étape : Effectuer la projection

- La commutation de la jointure et la restriction réduit considérablement le nombre de transferts de la mémoire secondaire vers la mémoire centrale, donc implique un gain de temps et donc une réduction du coût

# Optimisation des Requêtes

## *2 Questions*

- Comment passe-t-on d'une requête SQL (définie dans un langage déclaratif) à un programme manipulant les données ?

- Comment optimise-t-on une requête afin de l'exécuter efficacement pour trouver un résultat correct ?

# Optimisation des Requêtes

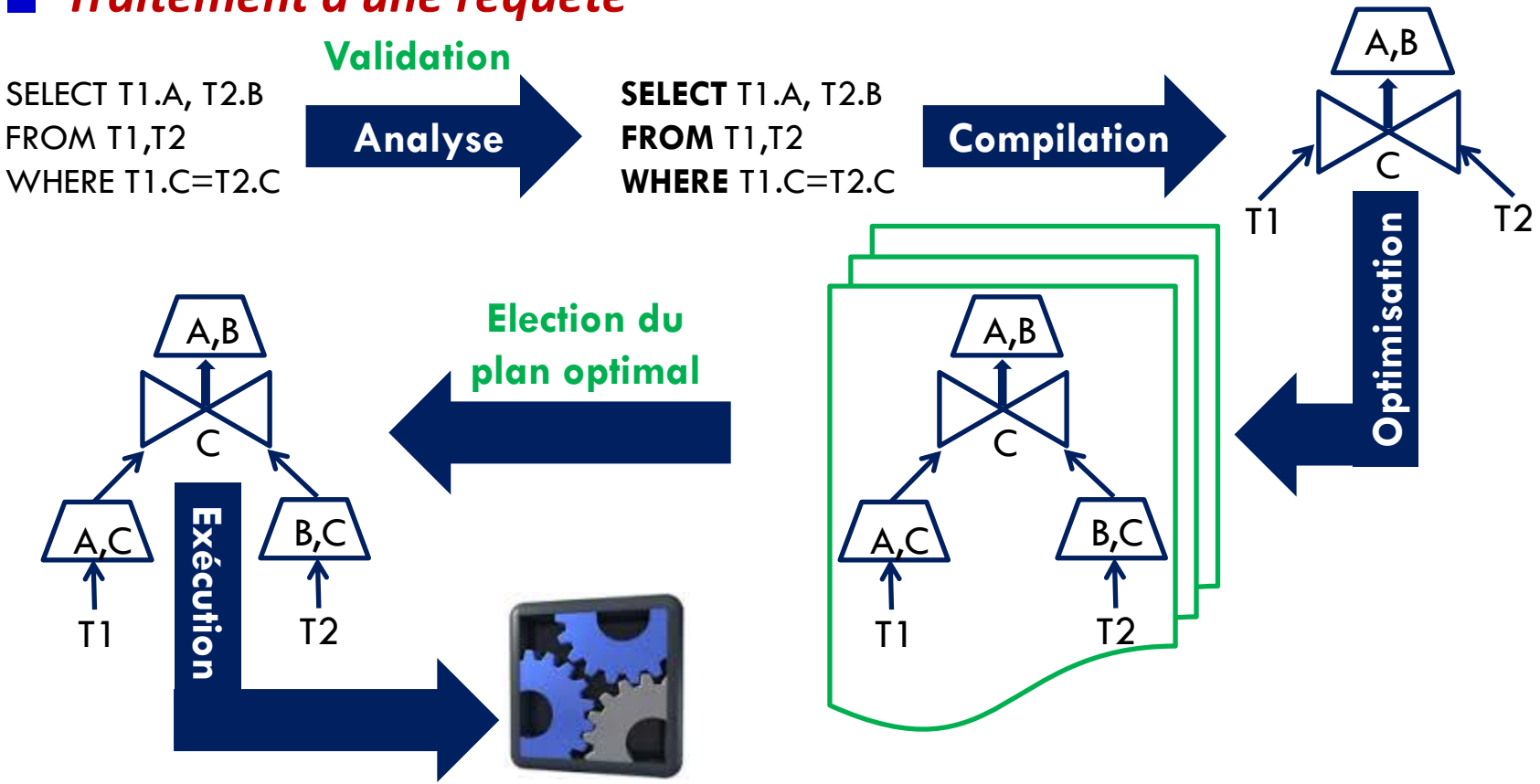
## *Traitement d'une requête*

Toute requête SQL est traitée en quatre étapes :

- **Analyse**
  - Validation de la syntaxe de la requête
- **Compilation**
  - Introduction de la requête en arbre algébrique
- **Optimisation**
  - Recherche de l'arbre algébrique optimal
- **Exécution**
  - Exécution du programme correspondant à l'arbre algébrique optimal

# Optimisation des Requêtes

- ***Traitement d'une requête***



# Optimisation des Requêtes

## *Analyse*

- **Analyse lexicale et syntaxique**

  - Validation par rapport à la syntaxe SQL
  - Existence des tables et colonnes
  - Compatibilité des types dans les prédicats
  - Vérification des privilèges
  - Décomposition en requêtes/sous-requêtes

# Optimisation des Requêtes

## *Compilation*

- **Un arbre algébrique est un arbre dont**

  - Les nœuds terminaux sont des relations
  - Les nœuds intermédiaires sont des opérateurs de l'AR
  - Le nœud racine est le résultat de la question
  - Les arcs sont les flux de données entre les opérateurs

# Optimisation des Requêtes

## *Compilation*

- **Règle de passage d'une requête SQL en AR**

- Elaboration d'un arbre de celle-ci indépendamment des valeurs et des chemins d'accès aux données
- SELECT …. : permet de définir une projection
- FROM … : permet de définir les feuilles de l'arbre
- WHERE ….: permet de définir
  - Pour les comparaisons attribut/valeur : des sélections
  - Pour les comparaisons attribut/attribut : des jointures

# Optimisation des Requêtes

## *Optimisation*

- **Trouver le meilleur plan d'exécution d'une requête**

  - Estimation des coûts d'exécution des plans de requête qui est en général différent
  - L'ordre des jointures est important
  - Trouver et choisir le plan d'exécution d'une requête dont le coût soit minimal (*le temps de réponse à la requête soit le plus rapide possible*). Qu'est-ce qui peut être optimiser dynamiquement ?
    - Les accès disques

  *Il est indispensable que le temps lié à l'optimisation soit négligeable par rapport au temps imparti à l'exécution*

# Optimisation des Requêtes

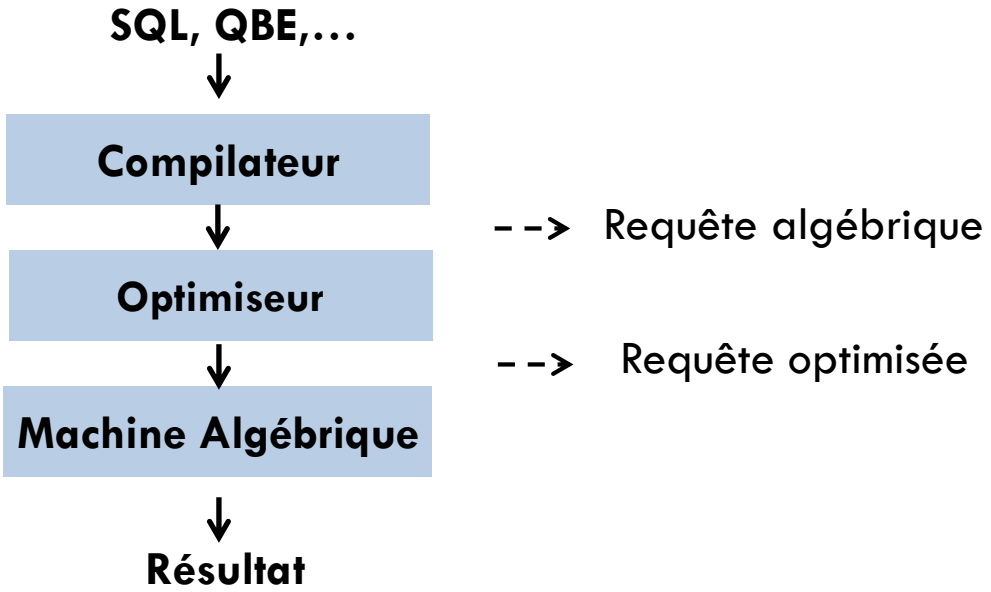
## *Exécution*

- **Le plan d'exécution est compilé et exécuté**

  - Chercher les informations dans les tables en passant par les chemins définis par le plan d'exécution
  - Application pour chaque opération du plan d 'exécution des programmes associés (Java, C, C++,...)
  - Extraire le résultat

# Optimisation des Requêtes

## *Traitement d'une requête*




- **Compilateur** : Traduction de la requête dans l'algèbre
- **Optimiseur** : Trouver un plan d'exécution tel que le coût d'exécution est minimal

# Optimisation des Requêtes

## *Principe*

- L'optimisation de requête est une opération dans laquelle plusieurs plans d'exécution d'une requête SQL sont examinés pour en sélectionner le meilleur
- L'estimation de leurs coûts dépend du temps d'exécution et du nombre de ressources utilisées pour y parvenir
- Les ressources coûteuses sont l'utilisation du processeur, la taille et la durée des tampons sur le disque dur, et les connexions entre les machines dans un système parallèle

**Deux types d'optimisation :**

- Plan d'exécution logique (PEL) qui dépend de l'algèbre relationnelle
- Plan d'exécution physique (PEP) qui tient compte des index et de la taille des données

# Optimisation des Requêtes

## *Principe*

- **2 problématiques importantes dans l'optimisation :**
  - Enumération des plans alternatifs pour une requête
  - Estimation des coûts de ces plans et choix de celui estimé être le moins cher

- **On doit se montrer pragmatique :**
  - Idéalement : Trouver le meilleur plan
  - Pratiquement : Éviter les pires plans !!

# Optimisation des Requêtes

## *Stratégie générale d'optimisation*

- Pour trouver les expressions équivalentes on doit utiliser les équivalences algébriques
- Réécriture et transformation des requêtes en fonction des propriétés des opérateurs pour obtenir la meilleure séquence d'opérations
- Réduire au plus tôt la taille et le nombre de tuples manipulé
  - Tuples moins nombreux (réduction de cardinalité) : grâce à la sélection
  - Tuples plus petits (réduction de degré) : grâce à la projection
- Traiter en une seule fois les opérations de restriction et projection sur une même relation

# Optimisation des Requêtes

## *Stratégie générale d'optimisation*

- Combiner les sélections avec un produit cartésien pour aboutir à une jointure
- Pré-traiter les relations avant d'effectuer la jointure soit en créant des index soit en effectuant le tri (*CREATE INDEX i-Fourniss ON Fournisseur (Nom ASC)* )
- Mémoriser les sous-expressions communes
- Réordonner les jointures, en fonction de la taille des relations manipulées
- Choisir des algorithmes de jointures efficaces selon les relations manipulées

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 1 : Regroupement ou cascade des projections**

$$\prod_{B1, B2, \ldots, Bp} (\prod_{A1, A2, \ldots, An} (R)) = \prod_{B1, B2, \ldots, Bp} (R)$$

avec $\{Bj\} \subset \{Ai\}$

Exemple : $\prod_{Nom} (\prod_{[Nom, Prenom, Adresse, Anee\_Bac]} (Etudiant) = \prod_{Nom} (Etudiant)$

- **Règle 2 : Regroupement ou cascade des restrictions**

$$\sigma_{P1(A1)} (\sigma_{P2(A2)} (R)) = \sigma_{[P1(A1) \wedge P2(A2)]} (R)$$

Exemple : $\sigma_{age>25} (\sigma_{genre='F'} (Etudiant)) = \sigma_{[age>25 \wedge genre='F']} (Etudiant)$

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 3 : Commutativité**

  $R1 \bowtie R2 = R2 \bowtie R1$

  Les opérateurs de produit, d'union et d'intersection sont aussi commutatifs, la différence ne l'est pas.

- **Règle 4 : Associativité**

  $(R1 \bowtie R2) \bowtie R3 = R1 \bowtie (R2 \bowtie R3) = (R1 \bowtie R3) \bowtie R2$

  Il existe **N!/2** arbres de jointure de N relations en appliquant les règles de commutativité et d'associativité

  L'union, l'intersection et la jointure sont toutes associatives. La différence ne l'est pas.

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 5 : Commutation des restrictions et projections**

**1ér Cas** : l'argument de restriction fait partie des attributs de projection

$$\prod_{A1, \ldots, An} (\sigma_{P(Ai)} (R)) = \sigma_{P(Ai)} (\prod_{A1, \ldots..,Ai\ldots, An} (R))$$

**2éme Cas** : Sinon

$$\prod_{A1, \ldots, An} (\sigma_{P(Ai)} (R)) = \prod_{A1, \ldots, An} (\sigma_{P(Ai)} (\prod_{A1, \ldots, An,Ai} (R))$$

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 6 : Commutation des restrictions avec l'union**

$$\sigma_{P(Ai)} (R \cup T) = \sigma_{P(Ai)} (R) \cup \sigma_{P(Ai)} (T)$$

- **Règle 7 : Commutation des restrictions avec la différence**

$$\sigma_{P(Ai)} ( \text{Minus } (R,T)) = \text{Minus } (\sigma_{P(Ai)} (R), \sigma_{P(Ai)} (T) )$$

- **Règle 8 : Commutation de la jointure et l'union**

$$(R \cup T) \bowtie S = (R \bowtie S) \cup (T \bowtie S)$$

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 9 : Commutation des projections avec l'union**

$$\prod_{A1, \ldots, An} (R \cup T) = \prod_{A1, \ldots, An} (R) \cup \prod_{A1, \ldots, An} (T)$$

- **Règle 10 : Commutation des projections et produit cartésien**

Soient $R (A_1, \ldots\ldots A_N)$ et $T (B_1, \ldots\ldots B_N)$

$$\prod_{A1,\ldots, Ap,B1,\ldots Bp} (R \otimes T) = \prod_{A1, \ldots, Ap} (R) \otimes \prod_{B1, \ldots, Bp} (T)$$

- **Règle 11 : Transformation de restrictions et produit cartésien en jointure**

$$\sigma_P (R \otimes T) = R \bowtie T$$

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 12 : Commutation des restrictions et produit cartésien**

**1ér Cas** : la restriction porte sur l'argument d'une seule relation : les deux relations n'ont pas d'attributs communs : R ($A_1$,….,$A_i$,….$A_N$) et T ($B_1$,….,$B_i$,…..$B_N$)

$\sigma_{P(Ai)} (R \otimes T) = \sigma_{P(Ai)} (R) \otimes T$

**2éme Cas** : la restriction porte sur deux arguments respectifs de R et T, les deux relations n'ont pas d'attributs communs : R ($A_1$,….,$A_i$,….$A_N$) et T ($B_1$,….,$B_i$,…..$B_N$)

$\sigma_{[P(Ai)\wedge P(Bj)]} (R \otimes T) = \sigma_{P(Ai)} (R) \otimes \sigma_{P(Bj)} (T)$

**3éme Cas** : la restriction porte sur deux arguments respectifs de R et T, les deux relations ont un attribut commun : R ($A_1$,….,$A_i$ ,…,X,….$A_N$) et T ($B_1$,….,X,…..$B_N$)

$\sigma_{[P(Ai)\wedge P(X)]} (R \otimes T) = \sigma_{P(X)} (\sigma_{P(Ai)} (R) \otimes T)$

# Optimisation des Requêtes

## *Règles de transformation des arbres*

- **Règle 13 : Commutation de la restriction à la jointure**

  $R(A_1,\ldots,A_i,\ldots,X,\ldots A_N)$ et $T(B_1,\ldots,X,\ldots B_N)$

  $\sigma_{p(Ai)}(R \bowtie T) = \sigma_{p(Ai)}(R) \bowtie S$

  *(1ér exemple fixant la problématique)*

- **Règle 14 : Commutation de la projection à la jointure**

  Soient $R(A_1,\ldots\ldots,X,\ldots\ldots A_N)$ et $T(B_1,\ldots\ldots,X,\ldots\ldots B_N)$

  $\prod_{[A1,\ldots,Ai,B1,\ldots Bj]}(R \bowtie T) = \prod_{[A1,\ldots,Ai,B1,\ldots Bj]}(\prod_{[A1,\ \ldots,\ Ai,\ X]}(R) \bowtie \prod_{[B1,\ \ldots,\ Bj,\ X]}(T))$

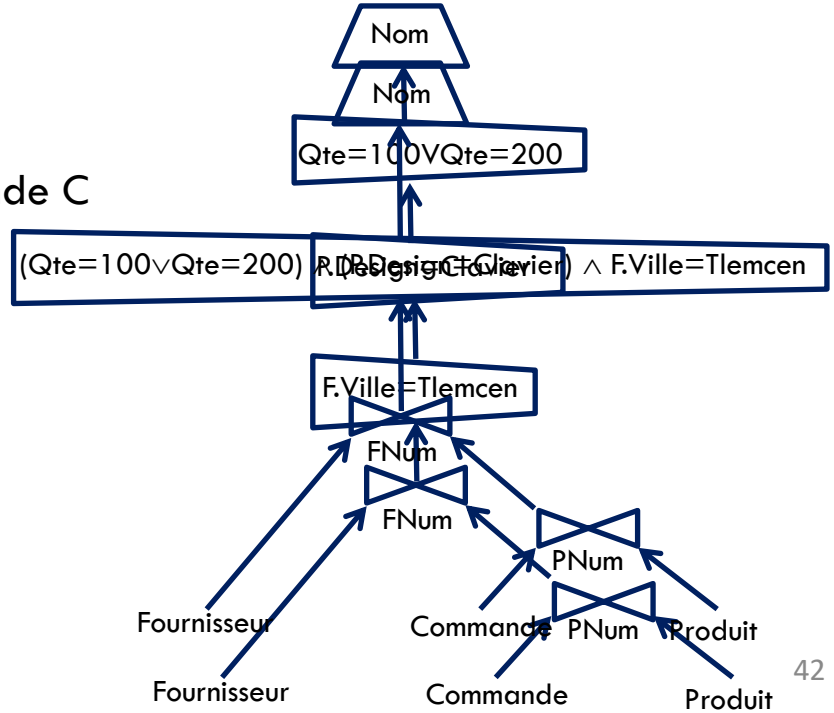
***Application de la règle*** *: Réduire au plus tôt la taille et le nombre de tuples manipulé en exécutant les sélections et les projections aussitôt que possible*

# Optimisation des Requêtes

## *Regroupement des opérations de restriction et projection*

- Exemple : Afficher les noms de fournisseurs résidant à Tlemcen à lesquels on a commandé 100 et 200 Claviers.

  - **SELECT** Nom
    **FROM** Fournisseur F, Produit P, Commande C
    **WHERE** F.FNum =C.FNum
    **AND** P.PNum= C.Pnum
    **AND** F.Ville='Tlemcen'
    **AND** P.Design='Clavier'
    **AND** (C.Qte=100 **OR** C.Qte=200);



# Optimisation des Requêtes

- ***Exécuter les sélections aussitôt que possible***

Appliquer ces règles (remplacer l'expression gauche avec la droite)

- $\sigma_{P(Ai)}\ (R \cup T) = \sigma_{P(Ai)}(R) \cup \sigma_{P(Ai)}(T)$
- $\sigma_{P(Ai)}\ (\text{Minus}\ (R,T)) = \text{Minus}\ (\sigma_{P(Ai)}(R), \sigma_{P(Ai)}(T))$
- $\sigma_{p(Ai)}\ (R \bowtie T) = \sigma_{p(Ai)}(R) \bowtie S$
- $\sigma_{P(Ai)}\ (R \otimes T) = \sigma_{P(Ai)}(R) \otimes T$

  Ou $\sigma_{[P(Ai)\wedge P(Bj)]}\ (R \otimes T) = \sigma_{P(Ai)}(R) \otimes \sigma_{P(Bj)}(T)$

  Ou $\sigma_{[P(Ai)\wedge P(X)]}\ (R \otimes T) = \sigma_{P(X)}(\sigma_{P(Ai)}(R) \otimes T)$

# Optimisation des Requêtes

## *Sous-expressions communes à une expression*

- Si une expression contient plusieurs fois la même sous-expression
- Si cette sous-expression peut être lue en moins de temps qu'il faut pour la calculer
- Matérialiser la sous-expression commune
  - CREATE **MATERIALISED** VIEW <Nom De La Vue> AS SELECT ……

# Optimisation des Requêtes

## *Réordonner les jointures en fonction de la taille des relations*

- **Exemple :** Afficher les noms de fournisseurs à lesquels on a commandé le produit CPU_I5
- **Hypothèses :** 40 fournisseurs et 500 commandes contenant 10 fois le produit CPU_I5
- **Sol 1:** $\prod_{Nom}(\sigma_{Design=CPU\_I5}(Produit \bowtie (Fournisseur \bowtie Commande)))$

  40 Four ⋈ 500 Comm → sur **20000** tuples composés, **500** sont sélectionnés

  Puis 1Prod ⋈ 500 tuples sélectionnés → sur **500** tuples, **10** sont sélectionnés finalement
- **Sol 2 :** $\prod_{Nom}(\sigma_{Design=CPU\_I5}((Produit \bowtie Commande) \bowtie Fournisseur))$

  1 Prod ⋈ 500 Comm → sur **500** tuples composés, **10** sont sélectionnés

  puis 10 tuples sélectionnés ⋈ 40 Four → sur **400** tuples, **10** sont sélectionnés finalement

**La deuxième stratégie est de loin la meilleure !**

# Optimisation des Requêtes

## *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(*,-,U). 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.

# Optimisation des Requêtes

## *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: Usine, Contrat, Ouvrier

  Si Cardinalité(Usine) <<< Cardinalité(Contrat) << Cardinalité(Ouvrier) alors quelle expression a de grandes chances d'être plus performante ?

  1. (Ouvrier ⋈ Usine) ⋈ Contrat
  2. (Usine ⋈ Contrat) ⋈ Ouvrier

- Optimisation basée sur les modèles de coût

# Optimisation des Requêtes

## *Optimisation basée sur les modèles de coût : 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 que contient un bloc
  - **Coût de lecture (ou écriture) d'une relation**= nombre de blocs à lire du disque vers la mémoire (ou à écrire de la mémoire vers le disque)

# Optimisation des Requêtes

## *Estimation du coût des opérations physiques*

- La performance d'une requête interne est évaluée en fonction :

| Métrique | Description |
|---|---|
| **TempsES** | Temps d'accès (lecture et écritures à la mémoire secondaire (MS) |
| **TempsUCT** | temps de traitement de l'UC (Souvent négligeable par rapport au 1er) |
| **TailleMC** | Espace requis en mémoire centrale |
| **TailleMS :** | Espace requis en mémoire secondaire |

- Dans les systèmes transactionnels :
  - Le principal souci est de pouvoir traiter les requêtes le plus rapidement possible
  - Le accès disque sont les opérations les plus coûteuses donc, la principale métrique de performance est **TempsES**

# Optimisation des Requêtes

## Statistiques d'évaluation d'une table T

| Statistique | Description |
|---|---|
| $N_T$ | Nombre de lignes de la table $T$ |
| $TailleLigne_T$ | La taille d'une ligne de la table $T$ |
| $FB_T$ | Facteur de blocage moyen de $T$<br>(nombre moyen de lignes contenues dans un bloc) |
| $B_T$ | Nombre de blocs de la table $T$<br>Estimation : $(NT / FBT)$ |
| $Card_T(col)$ | Nombre de valeurs distinctes (cardinalité) de la colonne col pour la table $T$<br>Ex : $CardT(sexe) = 2$ |
| $Min_T(col)$ | Valeur minimale que peut prendre une colonne $col$ |
| $Max_T(col)$ | Valeur maximale que peut prendre une colonne $col$ |

# Optimisation des Requêtes

## *Statistiques d'évaluation d'une expression de sélection*

- **$Sel_T$** : Taille ou Nombre de lignes distinctes de $T$ sélectionnées par l'expression de sélection

- **Facteur Sélectivité$_T$ (SF)** : Coût pour sélectionner une valeur spécifique d'un attribut équivalent au Pourcentage de lignes pour lesquelles la colonne **col** vérifie la condition de sélection [ 0 , 1 ]

  Estimation : **$SF_T$ (col) = $Sel_T$ / NT** ($SF_T$ (PK) = 1)

- Le Facteur de Sélectivité est important puisqu'il détermine le nombre de blocs à transférer (**TempsES**)

# Optimisation des Requêtes

## *Facteur de sélectivité*

- SF (A = valeur) = 1 / Card (A)
- SF(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
- SF(A < valeur) = (valeur - min(A)) / (max(A) - min(A))
- SF($valeur_1$ < A < $valeur_2$) = ($valeur_2$ - $valeur_1$) / (max(A) - min(A))
- SF(A IN liste valeurs) = (1 / Card (A)) * CARD(liste valeurs)
- SF(P et Q) = SF(P) * SF(Q)
- SF(P ou Q) = SF(P) + SF(Q) -SF(P) * SF(Q)
- SF(not P) = 1 - SF(P)

# Optimisation des Requêtes

## *Taille de jointure*

- Opérations coûteuses dans les requêtes relationnelles :
  - Jointure ($\bowtie$ )
  - Produit cartésien (X)

- Sel (R $\bowtie$ S)= SF(R $\bowtie$ S) * Sel (R) * Sel (S)

ou

- **SF(R $\bowtie$ S)** = Sel (R $\bowtie$ S) / (Sel (R) * Sel (S))

SF(R $\bowtie$ S) : Pourcentage de lignes pour lesquelles la colonne vérifie la condition de jointure

- SELECT * FROM R , S **SF = 1**

- SELECT * FROM R , S WHERE A = valeur **SF=1 / Card (A)**

# Optimisation des Requêtes

## *Coût du produit cartésien*

- **Hypothèses :**
  - R et S contiennent $n_R$ et $n_S$lignes
  - Un bloc disque peut contenir $b_R$lignes de R ou $b_S$ lignes de S
  - Nombre de blocs pour R et S : $k_R = \frac{n_R}{b_R}$/ et et $k_S = \frac{n_S}{b_S}$
  - m blocs peuvent être héberger en mémoire centrale
  - En gardant 1 bloc pour la lecture de S, on peut garder m-1 blocs de R en mémoire
  - Il faut lire $\frac{k_R}{m-1}$ fois m-1 blocs de R et $k_S$ blocs de S
  - Formule du coût (en nombre de blocs lus)

$\frac{k_R}{m-1}$ x (m-1+ $k_S$) lectures de bloc

# Optimisation des Requêtes

## *Coût du produit cartésien*

- Pour $k_R = 10$, $k_S = 2$ et $m = 6$ blocs en mémoire; il faut :

$$\frac{10}{6-1} \times (6 - 1 + 2) = 14 \text{ lectures de bloc}$$

| | | | | | |
|---|---|---|---|---|---|
| $S_1$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |
| $S_2$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |

→ Pour 7 lectures de blocs en gardant les premiers blocs de R pour chaque bloc de S

| | | | | | |
|---|---|---|---|---|---|
| $S_1$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ |
| $S_2$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ |

→ Pour 7 lectures de blocs en gardant les derniers blocs de R pour chaque bloc de S

# Optimisation des Requêtes

## *Exemple*

- Combien d'accès I/O ? Combien de temps de lecture ?
  - R : 10 000 tuples, 10 tuples/bloc → $k_R$=1000
  - S : 100 000 tuples, 10 tuples/bloc → $k_S$=1000
  - Nombre de blocs en mém

→ $\frac{1000}{99}$ x (99 + 1000) ≈

Et si on utilise 10% de R après une réduction ?

Réduction à 1110 accès soit 1,76 min (1000 lectures pour la sélection et 1100 pour le produit cartésien)

- Rappel des temps d'accès et
  - Mémoire : accès 10 ns, débit 2 Go/s
  - Disque dur : accès 5 ms, débit 50 Mo/s
- Nombre de blocs lus par secondes : $n_{bps}$= 20

→Temps de lecture = 9,25 minutes

# Optimisation des Requêtes

## *Procédures d'implantation prédéfinies des opérateurs*

- **Restriction**

  Implémentée différemment suivant que la formule de qualification fait référence à :

  - Une clé
  - Un attribut non clé mais indexé
  - Un attribut non clé non indexé

- **Algorithmes de jointure**

  - Jointures par boucles imbriquées (Nested Loop)
  - Jointure avec boucle indexée
  - Jointure par tri fusion (Sort-Join)
  - Jointure par hachage (Hash-Join)

# Optimisation des Requêtes

## *Algorithmes de jointure*

- **Jointure par boucles imbriquées**
  - Implémentée brutalement à l'aide de deux boucles imbriquées
  - Une boucle externe complète sur les occurrences de R et une boucle interne complète sur les occurrences de 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 l_R de R
        POUR chaque ligne l_S de S
                SI <Cond Joint> sur l_R et l_S est satisfaite
                        Produire la ligne concaténée à partir de l_R et l_S
                FIN SI
        FIN POUR
FIN POUR
```

# Optimisation des Requêtes

## *Algorithmes de jointure*

- **Jointure avec boucle indexée**
  - Même principe avec une seule boucle
  - S'il existe un index sur l'attribut de jointure, il ne sera pas nécessaire d'effectuer la deuxième boucle mais effectuer un accès direct au tuple (recherche indexée)

# Optimisation des Requêtes

## *Algorithmes de jointure*

- **Jointure par tri fusion**
  - Il sera possible de trier les deux relations suivant l'attribut de jointure
  - Le parcours des deux relations pourra être synchronisé et ne fournira qu'un seul parcours de données sans redondance (implémentation par fusion)
  - Plus efficace que les boucles imbriquées pour les grosses tables
  - Très efficace quand une des deux tables est petite (1, 2, 3 fois la taille de la mémoire)

**Principe**

1. Trier les deux tables sur les colonnes de jointure
2. Effectuer la fusion

**Complexité** : Tri qui coûte cher

# Optimisation des Requêtes

## *Algorithmes de jointure*

- **Jointure par tri fusion**

1. Trier R et S par tri externe et réécrire dans des fichiers temporaires
2. Lire le groupe de lignes GR(CR) de R pour la première valeur CR de clé de jointure
3. 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
```

# Optimisation des Requêtes

## *Algorithmes de jointure*

- **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


Fin