

# OPTIMISATION DE REQUETES :

---

BASES DE DONNÉES RELATIONNELLES

## INTRODUCTION

---

### PROBLEMATIQUE:

- Un des objectifs des bases de données est de restituer l'information dans des **délais acceptables**.
- Il s'agit de trouver la meilleure façon d'accéder et de traiter les données afin de répondre le plus rapidement possible à la requête: ceci passe par **l'évaluation de la requête** :
- 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 MAJ en MS

**L'optimisation consiste à réduire au maximum le volume de ces transferts**

- **Pour cela, le SGBD dispose d'un module d'évaluation de questions qui permet de choisir la meilleure stratégie possible d'exécution d'une expression relationnelle.**

### EXEMPLE fixant la problématique:

Nous travaillerons sur la base de données des fournisseurs avec les hypothèses suivantes :

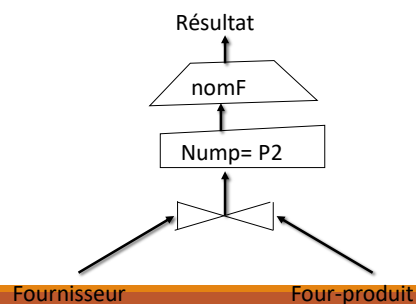
Cardinalité relation fournisseur = 100, cardinalité relation four-produit = 10000

**Soit la requête suivante** : « donner les noms des fournisseurs de la pièce P2 »

La requête SQL est la suivante :

```
SELECT nomf
FROM fournisseur, four-produit
Where nump = 'P2' and fournisseur.numf= four-produit.numf
```

Un arbre algébrique non optimisé de la requête est le suivant :



#### Evaluation de l'arbre non optimisé :

**1<sup>ère</sup> étape** : Jointure des relations fournisseur et four-produit :  
lecture des 10000 tuples de four-produit, puis lecture de chacun des 100 fournisseurs 10000 fois (soit 10000x100 accès disque au max), construction du résultat intermédiaire de jointure en MC(10000 tuples joints au minimum ). Il n'est pas sûr que celle-ci puisse être gardée en mémoire centrale.

**2<sup>ème</sup> étape** : restriction du résultat de l'étape précédente (10000x 50 accès au max). Les seuls tuples vérifiant num= $P_2$  seront gardés. Nous supposons qu'ils sont 50 et peuvent être gardés en mémoire.

**3<sup>ème</sup> étape** : projection sur le nom  
parcours des 50 tuples du résultat précédent afin de ne garder que nom

**Nous constatons la perte de temps en effectuant la jointure en premier du fait des lectures répétées.**

#### **Autre stratégie possible :**

**1<sup>ère</sup> étape** : effectuer la restriction de four-produit à num=  $P_2$   
lecture des 10000 tuples de four-produit pour ne garder en mémoire qu'une table de 50 tuples

**2<sup>ème</sup> étape** : effectuer la jointure du résultat précédent avec fournisseur  
cette étape entraîne l'extraction des 100 fournisseurs (100 accès disque). Le résultat contient 50 tuples.

**3<sup>ème</sup> étape** : effectuer la projection

**Ainsi 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. Nous verrons plus formellement cette stratégie dans les paragraphes suivants.**

## **PLAN**

- I. Typologie de requêtes**
- II. Les différentes phases de traitement d'une requête**
- III. Optimisation de requêtes**
  - i. Méthode syntaxique**
  - ii. Estimation de coûts**
  - iii. Techniques d'implémentation des opérateurs: cas de la jointure**

## **TYPOLOGIE DE REQUETES**

---

## Typologie des requêtes (I)

### ❑ Exemple d'une BD:

Ouvrier (NSS, Nom, Salaire, Spec, #N°U)

Usine (N°U, Site, Prod, #NSSChef)

Contrat(N°Cont, Fin, Cout, #NSS, #N°U)

### ❑ Classification des requêtes:

#### ➤ Requête singulière (Point query)

- Requête basée sur une seule relation et dont le prédicat de sélection est composé avec une ou plusieurs égalités de manière à ce que la réponse comprenne au plus un tuple.

Exemple:

```
SELECT * FROM Ouvrier WHERE NSS = 1233333;
```

## Typologie des requêtes (II)

#### ➤ Requête singulière multiple (multipoint query)

- Requête basée sur une seule relation dont le prédicat de sélection est composé d'une ou plusieurs égalités, et dont la réponse comprend plusieurs tuples

#### • Exemple:

```
SELECT * FROM Ouvrier WHERE Salaire > 1500 AND Spec = 'soudeur';
```

#### ➤ Requête d'intervalle (range query)

- Requête basée sur une seule relation et dont le prédicat comprend un test d'intervalle.

#### • Exemple:

```
SELECT * FROM Ouvrier WHERE Salaire > 1500 AND Salaire < 5000;
```

## Typologie des requêtes (III)

### ➤ Requête d'appartenance à un ensemble ordonné d'attributs Z

- Requête dont les conditions d'égalités doivent inclure obligatoirement une sous-chaîne définie selon l'ensemble Z

- Exemple:  $Z = \{NSS, Nom\}$

- `SELECT Nom, Salaire FROM Ouvrier WHERE NSS = 12345 AND Nom = 'Dupont';`

### ➤ Requête min-max

- Requête formulée sur une seule relation et dont le prédicat est composé d'égalités faisant référence à une valeur minimale ou maximale.

- Exemple: `SELECT Nom, Salaire FROM Ouvrier WHERE Salaire = max (SELECT salaire FROM Ouvrier);`

11

## Typologie des requêtes (IV)

### ➤ Requête de groupement (Grouping query)

- Requête dont la réponse est partitionnée par un ou plusieurs attributs

- Exemple:

- `SELECT N°U, AVG(Salaire) FROM Ouvrier GROUP BY N°U;`

### ➤ Requête de jointure (join query)

- Requête basée sur plusieurs relations.

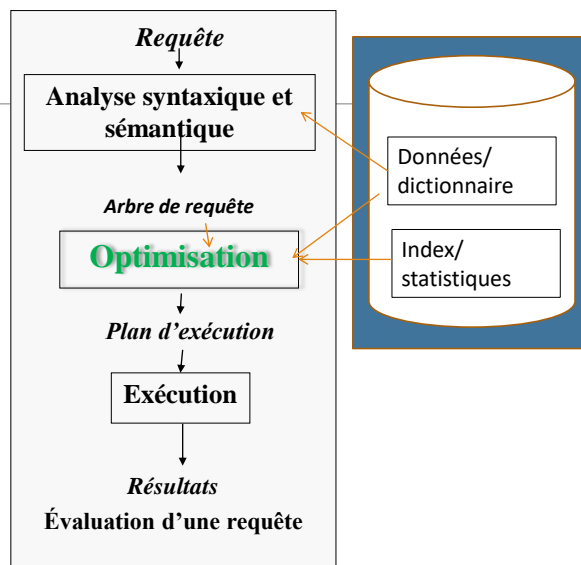
- Exemple:

- `SELECT NSS, Nom, Site FROM Ouvrier O, Usine U WHERE O.N°U = U.N°U;`

12

# Les différentes phases de traitement d'une requête

## Traitement de requête



**Analyse:** durant cette étape, la requête SQL est transformée en **arbre algébrique** ( si utilisation des opérateurs de l'algèbre relationnelle) ou en graphe de connexion (si utilisation du calcul relationnel). *On se place dans le cadre de l'arbre algébrique*

**Optimisation:** des règles et heuristiques sont appliquées sur l' arbre issu de la phase précédente, afin de déterminer l'ordonnancement optimal des opérations relationnelles, ainsi que l'utilisation des méthodes d'accès aux données qui permettront de générer un **plan d'exécution optimum**.

**Exécution** : c'est l'étape finale, qui consiste dans l'exécution du plan d'exécution afin d'élaborer ses **résultats**.

## Les différentes méthodes :

1. **Syntaxique** : transformation d'arbres de requêtes en utilisant des règles de transformation sur l'ordonnancement des opérateurs algébriques
2. **Estimation des coûts d'exécution** des diverses stratégies d'évaluation en se basant sur des informations statistiques et sur les méthodes d'accès
3. **Sémantique** : exploitation des contraintes d'intégrité

Ces méthodes sont non exclusives et souvent 1 et 2 sont combinées. Nous travaillerons sur les méthodes 1 et 2.

# OPTIMISATION DE REQUÊTES

---

METHODE SYNTAXIQUE

## Analyse de la requête

### Objectif :

étude **syntactique** de la question, c'est-à-dire vérification de sa correction et parfois une analyse **sémantique** est effectuée. Lors de cette étape, il y a vérification de la présence des attributs référencés avec les catalogues.

**L'analyse sémantique** consiste à vérifier si la question est bien formulée comme par exemple s'il y a oubli d'un opérateur ou s'il y a des contradictions (sélection avec deux critères  $>10$  et  $<10$  par exemple).

### Résultat : génération d'un arbre de requêtes.

Un arbre de requêtes est construit en utilisant une syntaxe abstraite associée aux opérateurs de l'algèbre relationnelle ou au calcul relationnel. Pour des raisons de commodité et afin de faciliter la compréhension, nous travaillerons sur une syntaxe proche de la notation utilisée dans l'algèbre relationnelle. L'arbre de requêtes sera alors un arbre algébrique.

**Un arbre algébrique** est un arbre dont :

- ☐ les nœuds terminaux sont des relations,
- ☐ les nœuds intermédiaires sont des opérateurs de l'algèbre relationnelle ,
- ☐ le nœud racine est le résultat de la question, les arcs sont les flux de données entre les opérateurs.

# OPTIMISATION DE REQUÊTES

---

ORDONNANCEMENT DES OPERATEURS

## **Méthode syntaxique basée sur l'ordonnancement des opérateurs**

L'ordonnancement des opérations relationnelles est complexe. Nous ne verrons dans cette partie qu'un ordonnancement basé sur une restructuration algébrique. L'optimisation dépend en grande partie de l'ordre d'exécution des opérateurs apparaissant dans l'arbre algébrique utilisé.

## **Stratégie générale d'une optimisation**

- Effectuer les opérations unaires de restriction et projection le plus tôt possible car elles réduisent respectivement le degré et la cardinalité d'une relation.
- Pré-traiter les relations avant d'effectuer la jointure soit en créant des index soit en effectuant le tri.
- Rechercher les sous expressions communes à plusieurs sous-arbres pour les pré-évaluer et stocker le résultat afin de le réutiliser.
- Traiter en une seule fois les opérations de restriction et projection sur une même relation.

Règles de transformation des arbres

Règles de regroupement

Règle 1 : regroupement des projections

$$\pi \left( \pi \left( R / A_1, A_2, \dots, A_n \right) / B_1, B_2, \dots, B_p \right) = \pi \left( R / B_1, B_2, \dots, B_p \right)$$

avec  $\{B_j\} \subset \{A_i\}$

Dans une séquence d'opérations de projection seule la dernière est nécessaire

$$\pi \left( \pi \text{ Employé} / \text{NSS, NOME, PRENOME} \right) / \text{NSS} = \pi \text{ Employé} / \text{NSS}$$

Règle 2 : regroupement des restrictions

$$\sigma \left( \sigma \left( R / X_i \theta a \right) / X_j \theta b \right) = \sigma \left( R / X_i \theta a \wedge X_j \theta b \right)$$


$$\sigma \left( \sigma \text{ Employé} / \text{Adresse} = \text{'Alger'} / \text{date-naissance} = \text{'21-10-2000'} \right) = \sigma \text{ Employé} / \text{Adresse} = \text{'Alger'} \wedge \text{date-naissance} = \text{'21-10-2000'}$$

Règles de transformation des arbres

Règles de commutativité

Règle 3 : Commutativité des jointures et produit cartésien


$R_1$



$R_2$


$=$

$R_2$




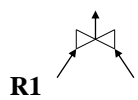
$R_1$

$\text{Ou}$




$\text{Ou}$





$=$



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

## Règles de transformation des arbres

### Règles de commutation ou distributivité

La distributivité d'un opérateur  $\Theta$  sur un opérateur  $\theta$  s'exprime comme suit :

$$\Theta (R1 \theta R2) = (\Theta R1) \theta (\Theta R2)$$

**Règle 4 :** Commutation des restrictions et projections

1<sup>er</sup> cas : l'argument de restriction fait partie des attributs de projection

$$\pi / (x_1, \dots, x_i) (\sigma (R / (x_p \theta a))) = \sigma / (x_p \theta a) (\pi (R / (x_1, \dots, x_p, \dots, x_i)))$$

$$\pi / \text{NSS, Adresse} (\sigma \text{ Employé / adresse='Alger'}) = \sigma / \text{adresse='Alger'} (\pi \text{ Employé / NSS, adresse})$$

2<sup>ème</sup> cas : Sinon

$$\pi / (x_1, \dots, x_i) (\sigma (R / (x_p \theta a))) = \pi / x_1, \dots, x_i (\sigma / x_p \theta a (\pi (R / (x_1, \dots, x_i, x_p))))$$

## Règles de transformation des arbres

**Règle 5 :** Commutation des restrictions avec l'union

$$\sigma (\text{Union} (R1, R2) / x_p \theta a) = \text{Union} (\sigma (R1 / x_p \theta a), \sigma (R2 / x_p \theta a))$$

**Règle 6 :** Commutation des restrictions avec la différence

$$\sigma (\text{Minus} (R1, R2) / x_p \theta a) = \text{Minus} (\sigma (R1 / x_p \theta a), \sigma (R2 / x_p \theta a))$$

## Règles de transformation des arbres

**Règle 7 :** Commutation des restrictions avec le produit cartésien

1<sup>er</sup> cas : la restriction porte sur l'argument d'une seule relation : les deux relations n'ont pas d'attributs communs :  $R1 ( \dots, Xi, \dots ), R2 ( \dots, Yj, \dots )$

$$\sigma (R1 \otimes R2) / (Xi \theta a) = \sigma (R1 / (Xi \theta a)) \otimes R2$$

2<sup>ème</sup> cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations n'ont pas d'attributs communs :  $R1 ( \dots, Xi, \dots ), R2 ( \dots, Yj, \dots )$

$$\sigma (R1 \otimes R2) / (Xi \theta a \wedge Yj \theta b) = \sigma (R1 / (Xi \theta a)) \otimes \sigma (R2 / (Yj \theta b))$$

3<sup>ème</sup> cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations ont un attribut commun  $Xp$  :  $R1 ( \dots, Xi, Xp, \dots ), R2 ( \dots, Xp, \dots )$

$$\sigma (R1 \otimes R2) / (Xi \theta a \wedge Xp \theta b) = \sigma ( \sigma (R1 / (Xi \theta a)) \otimes R2 ) / (Xp \theta b) )$$

## Règles de transformation des arbres

**Règle 8 :** Commutation des projections et produit cartésien

Soient  $R1 ( \dots, Xi, \dots )$  et  $R2 ( \dots, Yj, \dots )$

$$\pi ( (R1 \otimes R2) / X1, \dots, Xp, Y1, \dots, Yp ) = \pi (R1 / X1, \dots, Xp) \otimes \pi (R2 / Y1, \dots, Yp)$$

**Règle 9 :** commutation des projections avec l'union

$$\pi ( (R1 \text{ Union } R2) / X1, \dots, Xp ) = \pi (R1 / X1, \dots, Xp) \text{ Union } \pi (R2 / X1, \dots, Xp)$$

## Règles de transformation des arbres

### Règle 10 : Cas de la jointure naturelle

Une jointure naturelle de  $R1 (X1, \dots, Xi, \dots, Xn)$ ,  $R2 (Y1, \dots, Xi, \dots, Ym)$  est une restriction d'un produit cartésien à  $Xi = Xi$  suivie de la projection sur  $R1R2$

Commutation avec la projection :



$$\pi((R1 \bowtie R2) / X1 \dots Xp Y1 \dots Yq) = \pi(\pi(R1 / X1, \dots, Xp, Xi) \bowtie \pi(R2 / Y1, \dots, Yp, Xi) / X1 \dots Xp Y1 \dots Yq)$$

**Remarque :** la dernière projection n'est nécessaire que si  $Xi$  n'appartient pas à  $X1, \dots, Xp$  ou  $Y1, \dots, Yq$

Commutation avec la restriction :  $R1 ( \dots, Xi, \dots)$  et  $R2 ( \dots, Xi, \dots, Yj, \dots)$

$$\sigma((R1 \bowtie R2) / (Yj \theta b)) = R1 \bowtie (\sigma(R2 / (Yj \theta b)))$$

## Règles de transformation des arbres

A toutes ces règles peuvent être rajoutées des **règles d'associativité**.

L'associativité est définie comme suit :

$$A \Theta (B \Theta C) = (A \Theta B) \Theta C$$

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

## Algorithme simple de génération d'un plan de requête

Un algorithme simple d'utilisation des règles précédemment décrites afin de générer des plans de requêtes est le suivant : [Miranda p 246]

- Appliquer les règles 1 et 2 pour regrouper les restrictions et projections
- Faire descendre les restrictions le plus bas possible à l'aide de la règle R4 (pour passer une projection), de R5 et R6 (pour passer une union ou une différence) de R7 (pour passer un produit ou une jointure) et combiner éventuellement des restrictions avec R2
- Faire descendre les projections à travers les projections en utilisant la règle R1, à travers les restrictions ( R4 ), à travers les produits (R8), à travers les jointures (R10)
- Supprimer les projections redondantes :

Au niveau des feuilles de l'arbre de requêtes si tous les attributs sont cités

Au niveau des nœuds par l'utilisation de la règle de regroupement R1

**Plusieurs plans de requêtes peuvent être générés par transformations d'expressions. L'optimiseur devra choisir alors le moins coûteux. C'est l'étape d'estimation des coûts d'exécution qui permettra de choisir.**

# OPTIMISATION DE REQUETES

---

ESTIMATION DES COUTS D'EXECUTION

- Une solution simple pour choisir le meilleur plan d'exécution consiste à les générer tous et à estimer le coût d'exécution de chacun et choisir le moindre coût.
- Malheureusement cette démarche n'est pas très réalisable car :
  - le nombre de plans peut être très grand
  - la taille des résultats intermédiaires pris en compte est un surcoût.

Une solution simple consiste à utiliser des connaissances statistiques stockées, des connaissances sur les coûts d'implémentation des opérateurs et des connaissances sur les techniques d'indexation utilisées , afin de choisir la meilleure stratégie d'exécution.

## Explain plan - exemples

Rows	Execution Plan
------	----------------

12	SORT AGGREGATE
2	SORT GROUP BY
76563	NESTED LOOPS
76575	NESTED LOOPS
19	TABLE ACCESS FULL CN_PAYRUNS_ALL
76570	TABLE ACCESS BY INDEX ROWID CN_POSTING_DETAILS_ALL
76570	INDEX RANGE SCAN (object id 178321)
76563	TABLE ACCESS BY INDEX ROWID CN_PAYMENT_WORKSHEETS_ALL
11432983	INDEX RANGE SCAN (object id 186024)

#### UTILISATION DE DONNÉES STATISTIQUES

L'estimation du coût d'exécution utilisent les statistiques de la base de données mémorisées dans les catalogues. Nous rappelons que certaines informations comme la cardinalité des relations existent déjà dans les catalogues déjà vus.

Certaines informations complémentaires sont disponibles comme le nombre de pages occupées par une table, le nombre distinct de valeurs dans une colonne, le nombre de niveaux d'index , le nombre de pages de chaque niveau ...

#### UTILISATION DE PROCÉDURES D'IMPLEMENTATION DES OPÉRATEURS

Le programme d'optimisation va avoir à sa disposition un ensemble de procédures d'implémentation prédéfinies des opérateurs.

**Par exemple :**

- ☐ la restriction pourra être implémentée différemment suivant que la formule de qualification fait référence à une clé, ou à un attribut non clé mais indexé, ou tout simplement à un attribut non clé non indexé.
- ☐ La jointure de deux relations R1 et R2 peut être implémentée :
  - ☐ brutalement à l'aide de deux **boucles imbriquées** (une boucle externe complète sur les occurrences de R1 et une boucle interne complète sur les occurrences de R2),
  - ☐ ou 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**)
  - ☐ ou encore il sera possible de trier les deux relations suivant l'attribut de jointure, ainsi le parcours des deux relations pourra être synchronisé et ne fournira qu'un seul parcours de données sans redondance (implémentation par **tri- fusion**).

## RAPPEL Index

- 
- ❑ Les index sont des fichiers qui contiennent les paires attribut(s) d'accès → numéro page
  - ❑ Un index primaire est construit sur la clé primaire d'une relation
  - ❑ Un index secondaire est construit sur un attribut quelconque d'une relation

37

## Techniques d'indexation

- ❑ **Arbre B (B-tree)**
- ❑ **Hachage**
- ❑ **Bitmap**

38

## Recherche dans une table avec un index

- ❑ Exemple: `SELECT * FROM Etudiant WHERE Age = 24;`
- ❑ Un index sur Age permet d'obtenir rapidement les RIDs des tuples des employés de 24 ans.

### ❑ Requêtes conjonctives

`SELECT * FROM Employe WHERE Salaire = 2000 AND Ville = 'Laghouat';`

- Supposons l'existence de deux index; un sur Salaire et l'autre sur Ville.
- Le premier index fournit les RIDs satisfaisant la 1ère condition qui seront regroupés dans une table temporaire T1. Idem pour le deuxième index ~ T2
- Enfin l'intersection de T1 et T2 donne la réponse

39

## Les choix d'Oracle

- ❑ Par défaut l'index est un arbre B
- Dès qu'on utilise une commande PRIMARY KEY, Oracle crée un arbre B sur la clé primaire
- ❑ Arbre est stocké dans un segment d'index

40

## Exemples de création d'index

- ❑ Le nom d'un index est choisi par le DBA et peut être normalisé en le préfixant par `idx_` suivi du nom de l'attribut indexé et terminé par le nom de la table

- ❑ Exemple:

```
CREATE UNIQUE INDEX idx_NSS_Ouvrier ON Ouvrier(NSS);
```

```
CREATE UNIQUE INDEX idx_NSS_Salaire_Ouvrier ON Ouvrier(NSS, Salaire);
```

Suppression d'un index

```
DROP INDEX <Nom de l'index>;
```

```
DROP INDEX idx_NSS_Ouvrier;
```

41

## Guide d'utilisation des index en cours d'exploitation d'une BD

- ❑ Les index utiles sont ceux nécessaires à l'exploitation courante des données
- ❑ Supprimer les index lorsqu'il s'agit de traiter un flux important de transactions de type mise à jour sur une table. Ils sont d'abord supprimés, puis recréés au besoin après les mises à jour
- ❑ Une accélération de la jointure est possible par la création des index: indexer les attributs de jointure permettra un accès plus rapide

42

## Algorithmes de jointure

### ❑ Jointure sans index

- Jointure par boucles imbriquées (nested loop)
- Jointure par tri-fusion (sort-join)

### ❑ Jointure avec index

- Jointure avec boucles indexées

43

## Jointure par boucles imbriquées (JBI): Force brute

### ❑ $R \bowtie_F S$

#### • Principe:

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

44

## Jointure par boucles imbriquées

```
POUR chaque i := 1 à m                /*boucle externe */
  POUR chaque j := 1 à n                /*boucle interne*/
    SI = R[i].C = S[j].C alors
      ajouter le n-uplet joint R[i]*S[j] au résultat
    Fin Si
  finPOUR
FinPOUR
```

Remarques :

- Nécessite  $m \times n$  opérations de lecture
- Cardinalité :
  - Si jointure plusieurs vers 1 : card = au plus cardinalité de R ou S dont la clé étrangère intervient dans la jointure
  - Si jointure plusieurs à plusieurs la cardinalité dépend du nombre de valeurs distinctes de l'attribut de jointure C dans R ou S.
    - $m \times n / d_{CR}$  ou  $m \times n / d_{CS}$  suivant si on commence par R ou S

45

## Recherche indexée

- Il existe un index X sur l'attribut de jointure de la relation interne S.
- Différence avec l'algorithme force brute : pour chaque tuple de la relation externe R, l'accès aux tuples de la relation interne est direct via l'index et non pas un balayage systématique:
- Le nombre total de lectures de n-uplet pour les relations R et S est simplement la cardinalité du résultat de jointure :  $m + m \times n / d_{CS}$

Algorithme :

```
/* on suppose qu'il existe un index X sur S.C */
Pour i := 1 à m:                        /* boucle externe*/
  /* supposons qu'il y ait k entrées d'index X[1].. X[k] avec comme valeur
  d'attribut de jointure indexée = R[i].C */
  Pour j:= 1 à n;                        /* boucle interne */
    /* soit S[j] le n-uplet indexé par X[j] */
    Ajouter le n-uplet joint R[i]*S[j] au résultat
  fin
Fin
```

## Jointure par tri fusion

❑ Plus efficace que les boucles imbriquées pour les grosses tables

❑ Principe:

- Trier les deux tables sur l'attribut de jointure
- Effectuer la fusion

❑ Complexité: Le tri qui coûte cher

47

## Jointure par tri fusion

Trier R et S par tri externe 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

48

## **Jointure par hachage**

- ❑ **Très efficace quand une des deux tables est petite (1, 2, 3 fois la taille de la mémoire)**
- ❑ **Algorithme en option dans Oracle**

49

## **Principe de la jointure par hachage**

- 1- On hache la plus petite des deux tables en n fragments**
- 2- On hache la seconde table, avec la même fonction, en n autres fragments**
- 3- On réunit les fragments par paire, et on fait la jointure**

50