Université Constantine 2
جامعة قسنطينة 2

Développement Avancé d'Applications Web

– Chapitre 3 –

Optimisation des Requêtes

Dr Bouanaka Chafia

NTIC

chafia.bouanaka@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	TLSI	Licence 3	Génie Logiciel (GL)

Résumé

Objectifs du cours

- Comprendre la problématique de **la performance** en BD
- Connaitre **les classes de solutions** aux problèmes de performance
- Maitriser **les techniques de conception** permettant d'optimiser les performances d'une BD

Plan du Cours

- Motivation de l'optimisation des requêtes
- Notion de Plan d'exécution
- Approches d'optimisation des Requêtes
  - Sémantique
  - Syntaxique
  - Physique

Section 1:
Motivation de l'optimisation des requêtes

Motivation de l'optimisation

Exemple introductif

On considère le schéma de base de données suivant :

- etudiant (**cid**, nom)
- Inscrire (**cid, idcours**)
- Cours(**idcours**, nom_cours)

- Soient les requêtes SQL suivantes :
  - Retrouver les informations de l'étudiant Benali
  - Donner les noms des étudiants ayant un cid < 00112235
  - Donner les noms des étudiants qui sont inscrits au cours 'TABD'

Motivation de l'optimisation

Exemple introductif

On considère le schéma de base de données suivant :

- etudiant (**cid**, nom)
- Cours(**idcours**, nom_cours)
- Inscrire (**cid, idcours**)

- Soient les requêtes SQL suivantes :
  - Retrouver les informations de l'étudiant Benali
  - Donner les noms des étudiants ayant un cid < 00112235
  - Donner les noms des étudiants qui sont inscrits au cours 'TABD'

- Ces requêtes seront écrites en algèbre relationnelle comme suit :
  - $\sigma_{nom='Benali'}(etudiant)$
  - $\pi_{nom}(\sigma_{cid<00112235}(etudiant))$
  - $\pi_{nom}(\sigma_{nom\_cours='TABD'}((etudiant \bowtie_{cid} inscrit) \bowtie_{idcours} cours))$

Motivation de l'optimisation

Exemple introductif

Pour exécuter une requête SQL, ou évaluer une expression algébrique, différents choix doivent être fait :

- Plusieurs solutions sont possibles pour évaluer ces expressions :
  $\pi_{nom}(\sigma_{nomcours=TABD}((etudiant \bowtie_{cid} inscrit) \bowtie_{idcours} cours))$
  $\pi_{nom}((etudiant \bowtie_{cid} (inscrit \bowtie_{idcours} (\sigma_{nomscours=TABD}(cours)))$

- Plusieurs options d'accès aux données sont possibles pour évaluer un opérateur donné :
  - $\Sigma_{nom = 'Benali}(etudiant)$
    - Parcours séquentiel du fichier **etudiant**
    - Utilisation d'un indexe secondaire sur **etudiant.nom**

- Plusieurs plans d'exécution de ces requêtes sont possibles :

- Plusieurs chemins d'accès :
  - Différents chemins d'accès aux enregistrements

- La réponse à ces questions est faite par le moteur de requêtes d'un SGBD
- Il définit un **plan d'exécution**

Motivation de l'optimisation

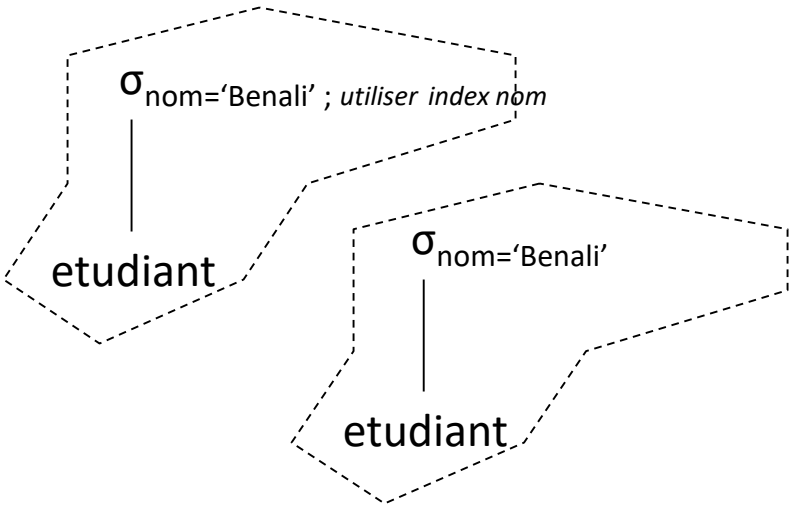
Exemple introductif

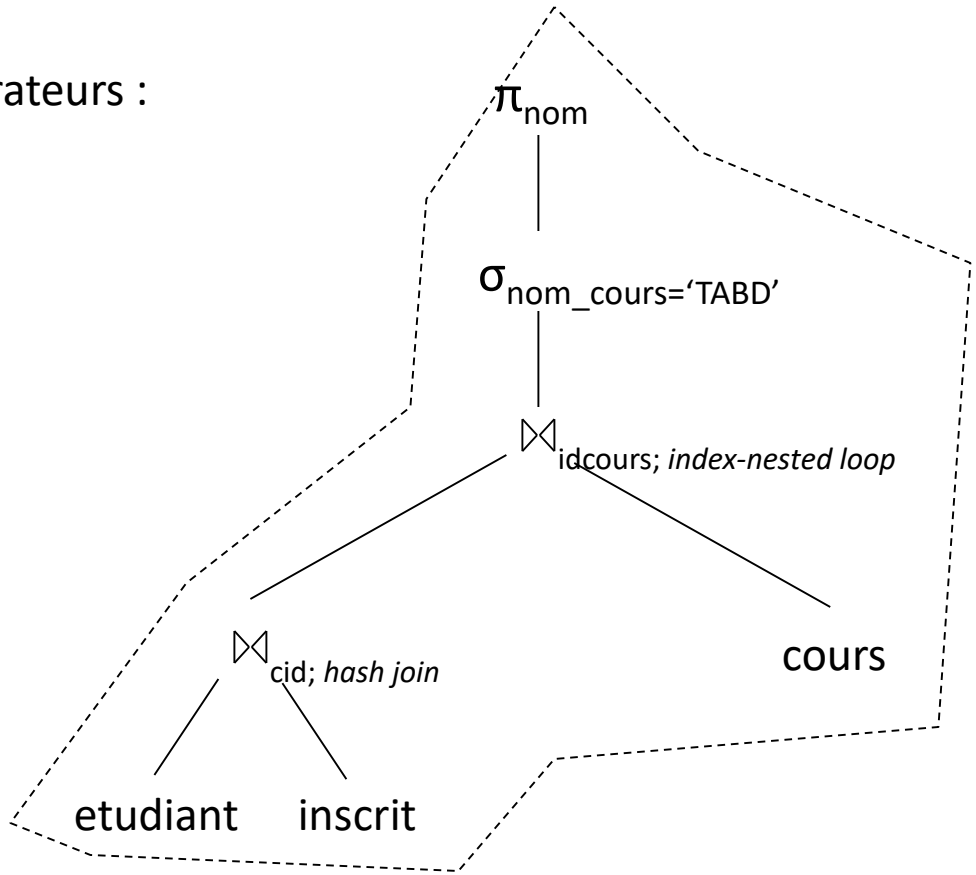
Plan d'exécution : sert à

- Spécifier le chemin d'accès à adopter
- Spécifier l'algorithme à utiliser pour évaluer chaque opérateur
- Spécifier le principe d'exécution des opérateurs :
  - ✓ matérialisation,
  - ✓ pipeline

Optimisation:

- Estimer le coût de chaque plan
- Sélectionner le plan le moins cher





Motivation de l'optimisation

Objectifs de l'optimisation

- Trouver **un plan d'exécution** permettant d'améliorer le **temps de traitement** de la requête.
- **Le temps de traitement** est calculé sur la base de :
  1. Le volume de données manipulées, en considérant certains paramètres :
     - La taille ou la cardinalité des relations (Le nombres de tuples)
     - La largeur d'un tuple d'une relation(Le nombre d'attributs de la relation)
     - Le nombre de valeurs possibles pour un constituant
     - La longueur d'un constituant
  2. Le temps d'exécution de l'algorithme d'implémentation des opérations de base

Motivation de l'optimisation

Objectif de l'optimisation

Constat : Afin d'optimiser une requête, il est nécessaire de

réduire la taille des données

Comment :

- en filtrant les tuples par des **restrictions**
- en les simplifiant par des **projections**

- Il est alors nécessaire de définir des règles permettant de transformer les requêtes afin de les optimiser

Section 2:
Notion de plan d'exécution

Notion de Plan d'exécution

Définition

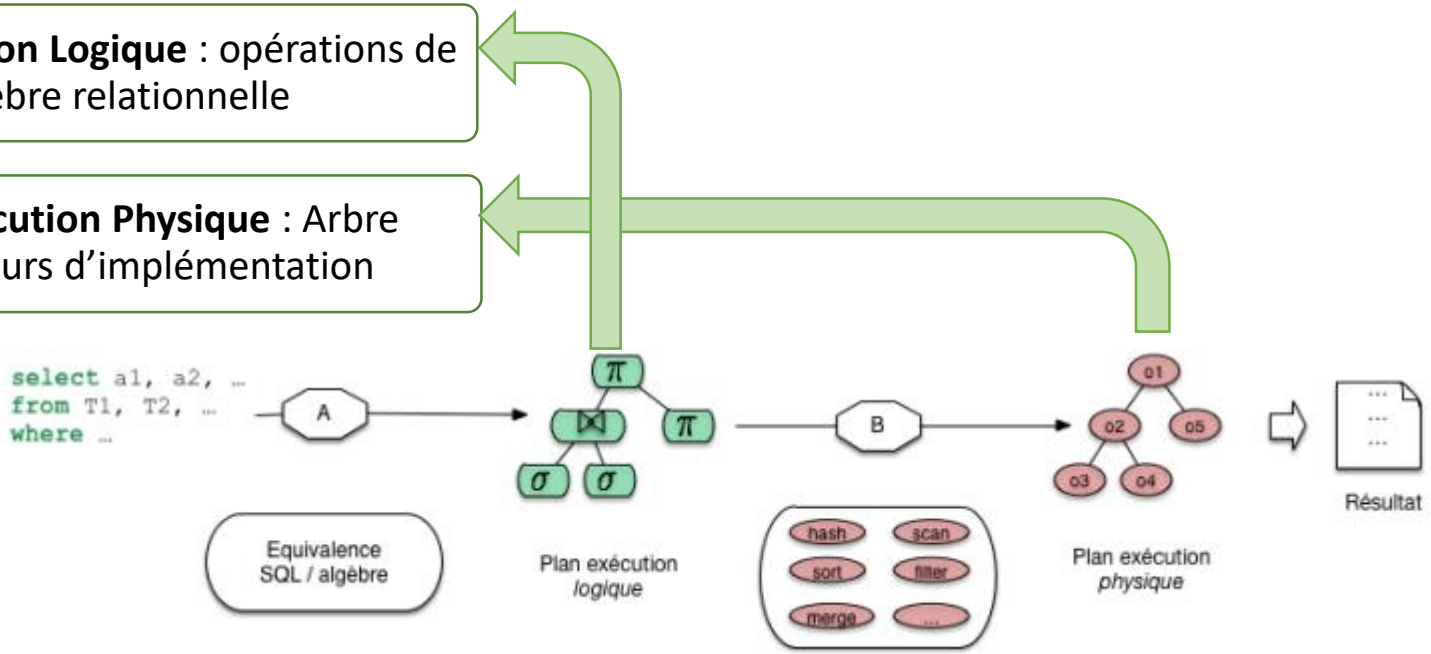
- Une requête SQL est **déclarative**.
- Elle ne précise pas comment calculer le résultat
- Besoin d'une forme **opératoire** : un programme
- Dans un SGBDR, le programme qui exécute une requête est appelé **Plan d'exécution**
- Ce plan est un arbre constitué des opérateurs de base

Notion de Plan d'exécution

Types de Plans

Types de Plan d'exécution

- **Plan d'exécution Logique** : opérations de l'algèbre relationnelle
- **Plan d'exécution Physique** : Arbre d'opérateurs d'implémentation



Important

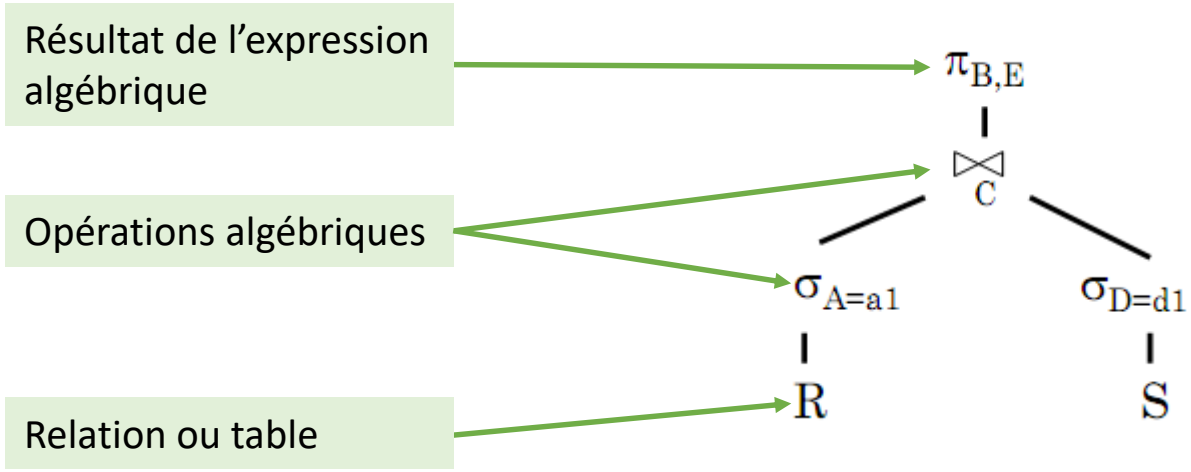
- **Chaque SGBD propose un catalogue d'opérateurs ou algorithmes d'implémentations des opérations de base de l'algèbre relationnelle**

Notion de Plan d'exécution

Plan d'exécution Logique

Définition

- Un **Plan d'Exécution Logique** de requête est un arbre dont les noeuds sont des opérateurs de l'algèbre relationnelle
- Un plan d'exécution logique est **une structure de données arborescente :**
- **Feuilles :** relations ou tables utilisées dans la requête
- **Nœuds intermédiaires :** opérations algébriques
- **Nœud racine :** dernière opération algébrique avant le retour du résultat



Notion de Plan d'exécution

Plan d'exécution Physique

Définition

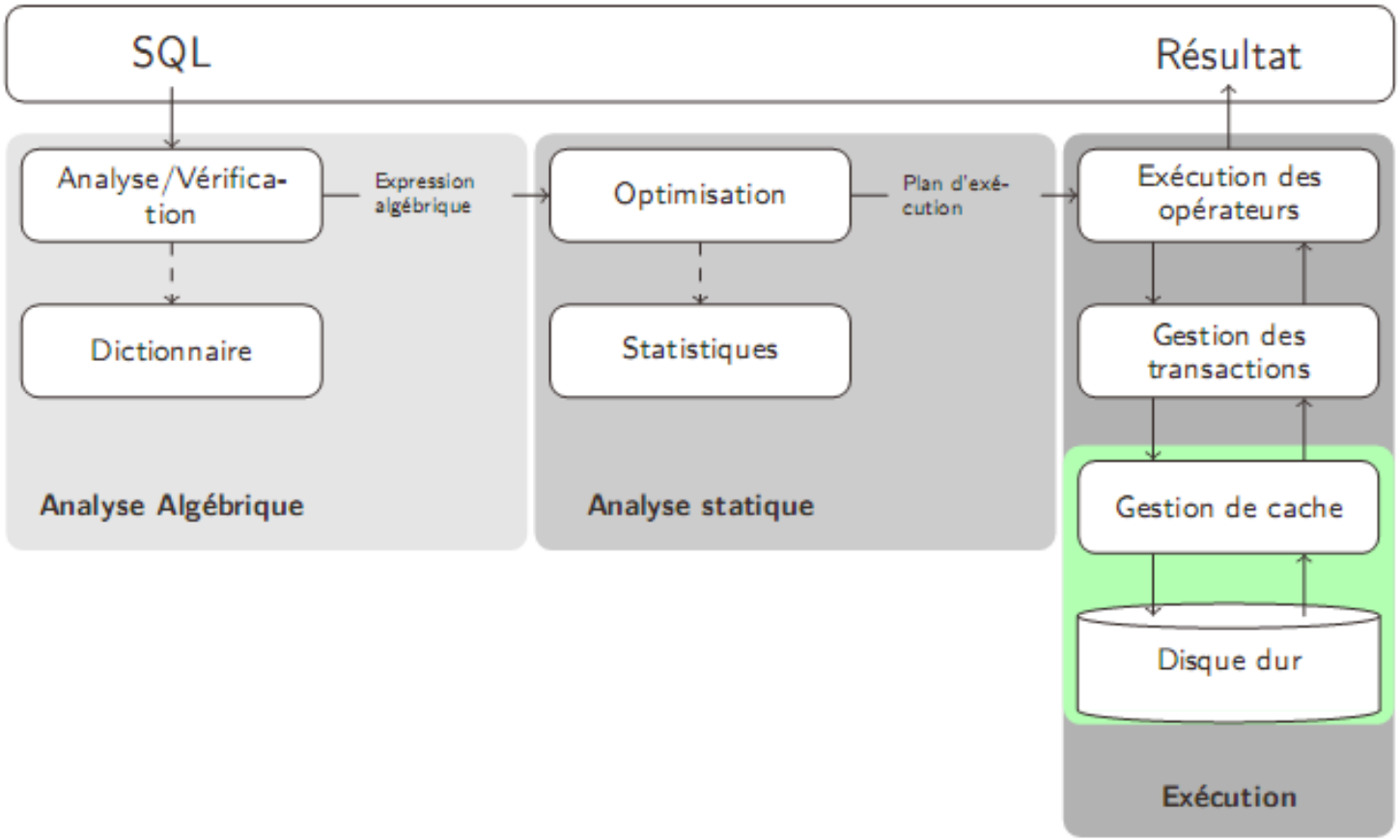
- Un **Plan d'Exécution Physique** est un plan d'exécution logique annoté par les algorithmes d'implémentation des opérateurs de base

Section 3 :
Approches d'optimisation

Approches d'optimisation

Etapes d'exécution d'une requête SQL

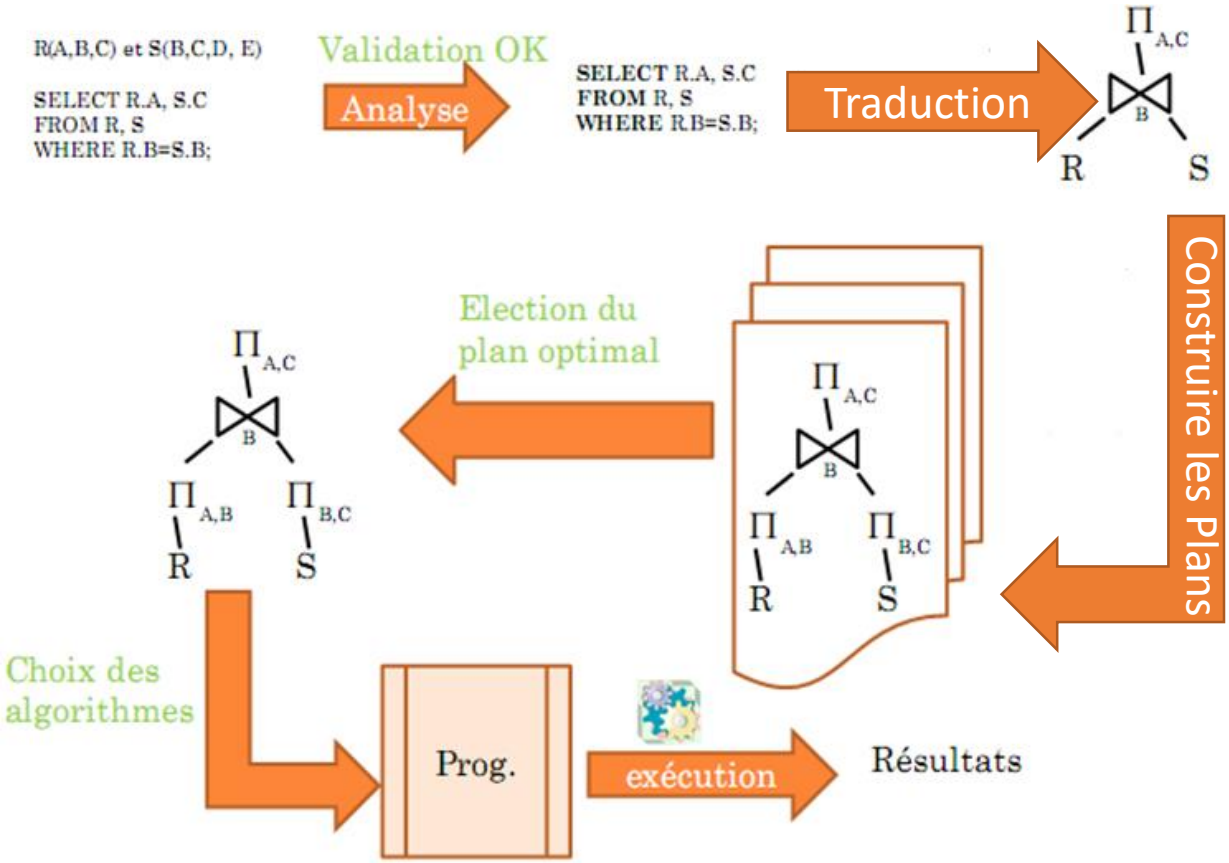
Chaque requête SQL passe par un ensemble d'étapes avant d'être exécutée et les résultats retournés à l'utilisateur. Ces étapes sont représentés dans la figure ci-dessous



Approches d'optimisation

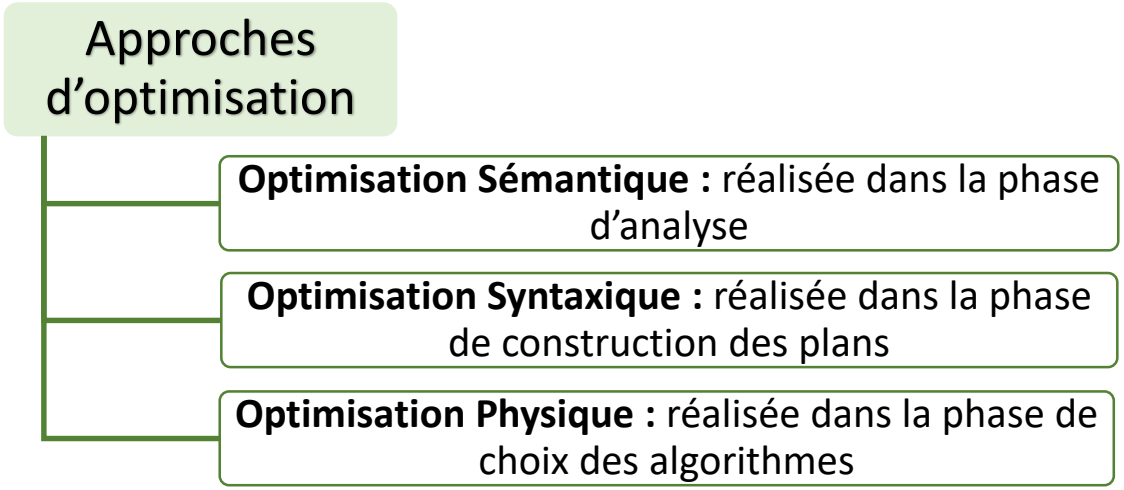
Etapes d'exécution d'une requête SQL

Les étapes d'analyse et d'optimisation sont très déterminantes dans l'exécution de la requête, elles impliquent un ensemble de taches comme illustrer dans la figure ci-dessous



Approches d'optimisation

Types d'optimisation



Approches d'optimisation

Optimisation Sémantique

Définition

- L'optimisation sémantique vise à détecter les contradictions dans une requête SQL et éviter ainsi son exécution

Exemple

```
SELECT *
From Personne
Where Age < 10 AND Age > 25
```

Optimisation Syntaxique

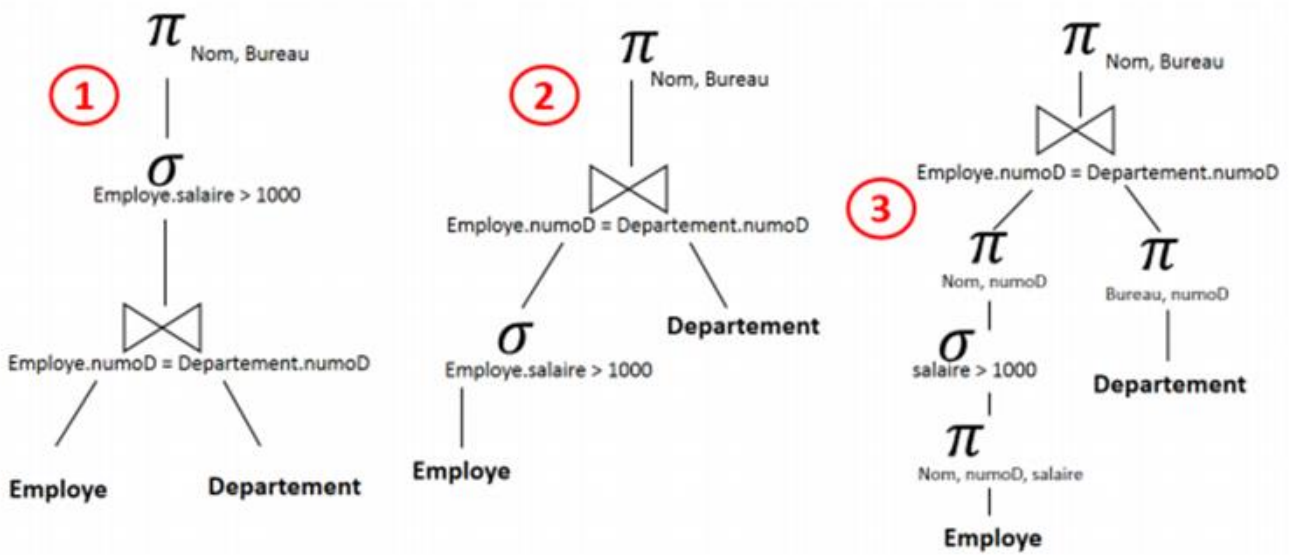
Approches d'optimisation

Optimisation Syntaxique

Définition

- Permet de réorganiser les différentes opérations relationnelles élémentaires d'une requête afin de réduire la quantité des données manipulées

$$\pi_{Nom,Bureau}\,(\sigma_{E.salaire>1000}(Employe \bowtie_{E.numD=D.numD} Departement))$$



Approches d'optimisation

Optimisation Syntaxique

L'optimisation syntaxique est réalisée en quatre étapes :

1. Traduire la requête SQL en une expression algébrique
2. Construire l'arbre algébrique
3. Générer les plans d'exécution par application des règles de transformation
4. Choisir le meilleur plan, c.-à-d., le plan le moins coûteux

Optimisation Syntaxique

ETAPE 1 :

Traduction de la requête SQL en Algèbre Relationnelle

Optimisation Syntaxique

L'étape de traduction consiste à trouver une expression de l'algèbre relationnelle équivalente à la requête SQL

Rappel Algèbre Relationnelle

La projection π : $\pi_{Nom,adresse}(Client)$

La sélection σ: $\sigma_{adresse=Bejaia}(Client)$

Le produit cartésien $\times$: $R \times S$

La jointure $\bowtie$: $Client \bowtie_{numero=no\_client} Vente$

L'union $\cup$: $R \cup S$

La différence $-$: $R - S$

Intersection, division, ...

Optimisation Syntaxique

Exemple

Soit la requête:

```
SELECT Nom, Bureau
FROM Employe, Departement
WHERE Employe.numoD = Departement.numoD
AND Employe.salaire > 1000
```

Traduction en Algèbre Relationnelle

$$\pi_{Nom,Bureau}\left(\sigma_{Employe.salaire>1000}\left(Employe \bowtie_{Employe.numoD=Departement.numoD} Departement\right)\right)$$

Optimisation Syntaxique

ETAPE 2 :

Tracer l'arbre algébrique

Optimisation Syntaxique

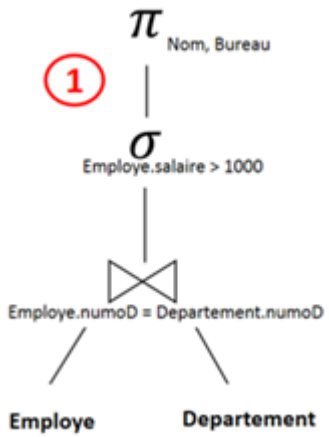
Définition

L'arbre algébrique est **une structure de données arborescente :**

- **Feuilles :** relations ou tables utilisées dans la requête
- **Nœuds intermédiaires :** opérations algébriques
- **Nœud racine :** dernière opération algébrique avant le retour du résultat

Exemple :

$$\pi_{Nom,Bureau}\left(\sigma_{Employe.salaire>1000}\left(Employe \bowtie_{Employe.numoD=Departement.numoD} Departement\right)\right)$$



ETAPE 3 :

Déduire les Plans d'exécution

Optimisation Syntaxique

- La déduction des plans d'exécution logique consiste à appliquer un ensemble de règles de transformation pour obtenir des expressions algébriques équivalentes
- L'objectif est de réorganiser les opérations algébriques afin réduire la tailles des données manipulées et obtenir des expressions **optimisées**

Principe de l'optimisation syntaxique:

Le principe générale repose sur le constat suivant :

- Les **opérateurs unaires** produisent des tables plus petites que la table d'origine
- Les **opérateurs binaires** produisent des table plus grandes que la table d'origine.
- En particulier, les **produits cartésiens** et les **jointures** accroissent la taille des tables intermédiaires.

Idée → Supprimer un maximum de lignes et de colonnes avant de faire les jointures. De plus faire les jointures avant les produits cartésiens.

Règles de Transformation :

Règle 1 : Commutativité et associativité de $\bowtie$ et $\times$

$$E1 \bowtie E2 = E2 \bowtie E1$$
$$(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$$

$\Rightarrow$ Si $(E1 \bowtie E2)$ est plus grand que $(E2 \bowtie E3)$, choisir $E1 \bowtie (E2 \bowtie E3)$

$$E1 \times E2 = E2 \times E1$$
$$(E1 \times E2) \times E3 = E1 \times (E2 \times E3)$$

Règle 2 : Regroupement/Séparation des projections

$$\pi_{A_1,A_2,\dots,A_n}\left(\pi_{B_1,B_2,\dots,B_n}(E)\right) = \pi_{A_1,A_2,\dots,A_n,B_1,B_2,\dots,B_n}(E)$$

Règle 3 : Regroupement/Séparation des restrictions

$$\sigma_{F_1}\left(\sigma_{F_2}(E)\right) = \sigma_{F_1 \wedge F_2}(E)$$

Règles de Transformation :

Règle 4 : Distributivité de la restriction sur les opérateurs binaires (x, ∪, −, ⋈)

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

Règle 5 : Commutativité de la restriction avec la projection

Si F ne porte que sur $A_1, \dots, A_n$,

$$\pi_{A_1,\dots,A_n}\left(\sigma_F(E)\right) = \sigma_F\left(\pi_{A_1,\dots,A_n}(E)\right)$$

Si F porte aussi sur $B_1, \dots, B_m$,

$$\pi_{A_1,\dots,A_n}\left(\sigma_F(E)\right) = \pi_{A_1,\dots,A_n}\left(\sigma_F\left(\pi_{A_1,\dots,A_n,B_1,\dots,B_m}(E)\right)\right)$$

Règle 4 : Distributivité de la projection sur les opérateurs binaires (x, ∪, −, ⋈)

$$\pi_F(E_1\ op\ E_2) = \pi_F\ (E_1)\ op\ \sigma_F\ (E_2)$$

Optimisation Syntaxique

Principe de l'optimisation:

L'optimisation intuitive se résume à :

1. Faire toutes les restrictions spécifiques pour limiter le nombre de tuples.
2. Faire toutes les projections mono-tables possibles pour limiter le nombre d'attributs.
3. Faire les jointures et après chaque jointure, les projections possibles.

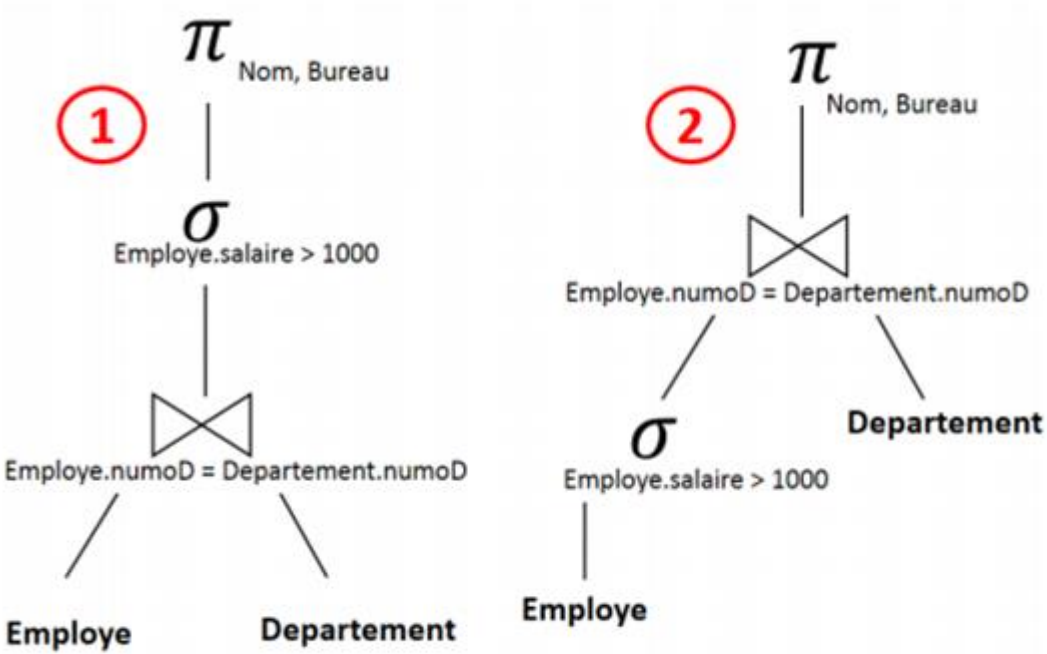
Algorithme d'optimisation:

En utilisant les règles de transformation, on obtient l'algorithme suivant :

1. Séparer les restrictions comportant plusieurs prédicats (**R3**).
2. Faire descendre les restrictions le plus bas possible (**R4, R5**)
3. Regrouper les restrictions successives portant sur une même relation (**R3**)
4. Séparer les projections comportant plusieurs prédicats (**R2**).
5. Faire descendre les projections le plus bas possibles (**R4, R6**)
6. Regrouper les projections successives (**R2**)

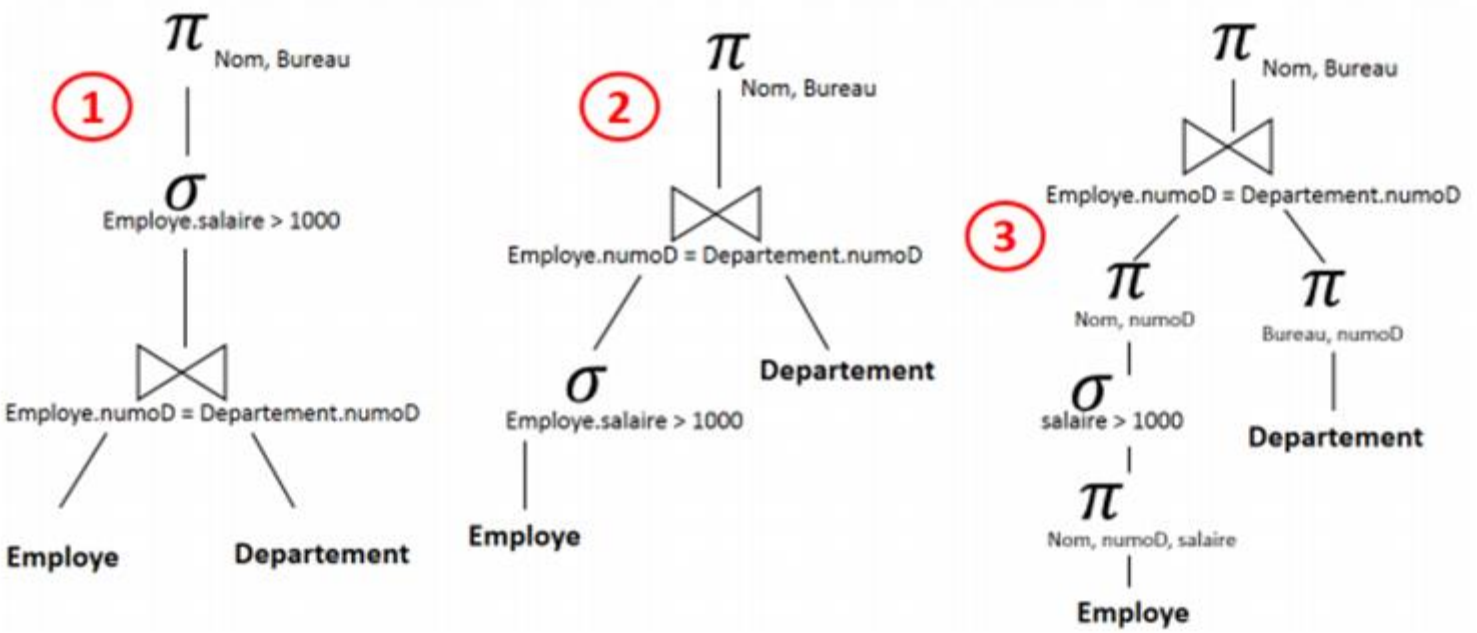
Etape 3 : Exemple

$$\pi_{Nom,Bureau}\left(\sigma_{Employe.salaire>1000}\left(Employe \bowtie_{Employe.numoD=Departement.numoD} Departement\right)\right)$$



Etape 3 : Exemple

$$\pi_{Nom,Bureau}\left(\sigma_{Employe.salaire>1000}\left(Employe \bowtie_{Employe.numoD=Departement.numoD} Departement\right)\right)$$



Optimisation Physique

Optimisation Physique

Principe

Définition

Prend en compte :

- les index,
- les statistiques (taille des tables et sélectivité des restrictions),
- la gestion de la mémoire cache et les principes algorithmiques classiques de l'accès aux données.

Important

- **L'optimisation physique se base sur la construction du plan d'exécution physique(PEP)**

Définition : Plan d'exécution physique

Un plan d'exécution Physique (**PEL**) est un arbre constitué **d'opérateurs** issus d'un « catalogue » propre à chaque SGBD et s'échangeant des **flux données**

Caractéristiques des opérateurs

- ont une forme générique (itérateurs) ;
- fournissent une tâche spécialisée
- peuvent être **bloquants** ou **non bloquants**.

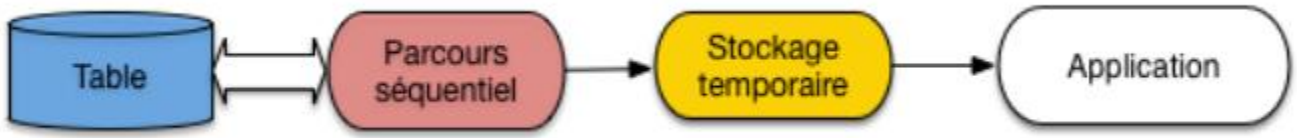
Modes d'exécution d'un opérateur

Il existe deux modes d'exécution :

- Matérialisation
- Pipeline

Le mode Matérialisation :

Chaque opérateur calcule son résultat ensuite le transmet



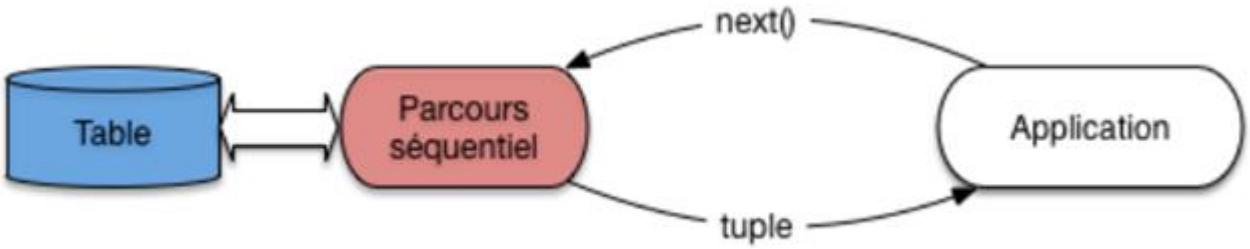
Inconvénients :

- Consomme de la mémoire
- Problème de latence

Modes d'exécution d'un opérateur

Le mode Pipeline :

Le résultat est produit à la demande



Avantages :

- On n'a plus besoin du stockage intermédiaire
- Latence minimale

Remarque

- **Certains opérateurs sont bloquants (Sum, min, max, Count..) : pas de pipeline**

Modes d'exécution d'un opérateur

Opérateurs Bloquants :

- Parfois il n'est pas possible d'éviter le **calcul complet** de l'une des opérations avant de continuer.
- L'opérateur est alors ***bloquant :*** le résultat doit être entièrement produit (et matérialisé en cache ou écrit sur disque) avant de démarrer l'opération suivante.

Exemples d'opérateurs bloquants :

- le tri (**order by**);
- la recherche d'un maximum ou d'un minimum (**max, min**);
- l'élimination des doublons (**distinct**);
- le calcul d'une moyenne ou d'une somme (**sum, avg**);
- un partitionnement (**group by**)

Définition : Opérateur

Définition

Tout opérateur est implanté sous forme d'un itérateur qui réalise les trois fonctions suivantes :

- **open** : initialise les ressources et positionne le curseur des enregistrements;
- **next** : ramène l'enregistrement courant et se place sur l'enregistrement suivant ;
- **close** : libère les ressources ;

Remarque

- **Les itérateurs peuvent être placés en pipeline :**
  - **Un itérateur consomme des tuples d'autres itérateurs source.**
  - **Un itérateur produit des tuples pour un autre itérateur (ou pour l'application).**

Types d'opérateurs

Les opérateurs d'accès :

On peut distinguer tout d'abord les opérateurs d'accès:

- le parcours séquentiel d'une table : **FullScan**,
- le parcours d'index : **IndexScan**,
- l'accès direct à un enregistrement par son adresse, **DirectAccess**, nécessairement combiné avec le précédent.

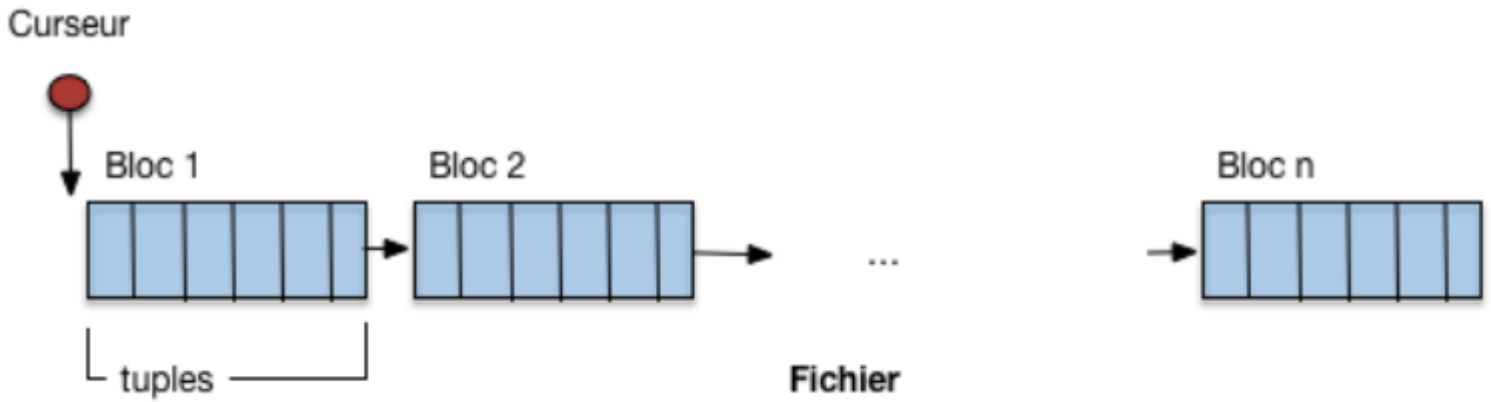
Types d'opérateurs

Les opérateurs de manipulation :

- la sélection : **Filter**,
- la projection : **Project**,
- le tri : **Sort**,
- la fusion de deux listes : **Merge**,
- la jointure par boucles imbriquées indexées, **IndexedNestedLoop**.

Parcours Séquentiel : FullScan

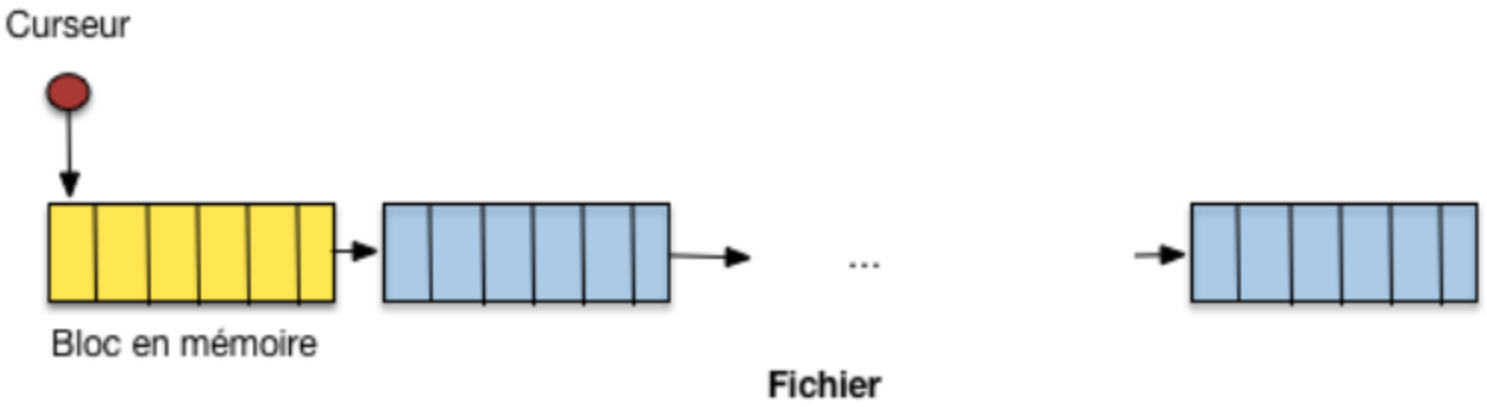
Au moment du open(), le curseur est positionné avant le premier tuple.



open() désigne la phase d'initialisation de l'opérateur.

Parcours Séquentiel : FullScan

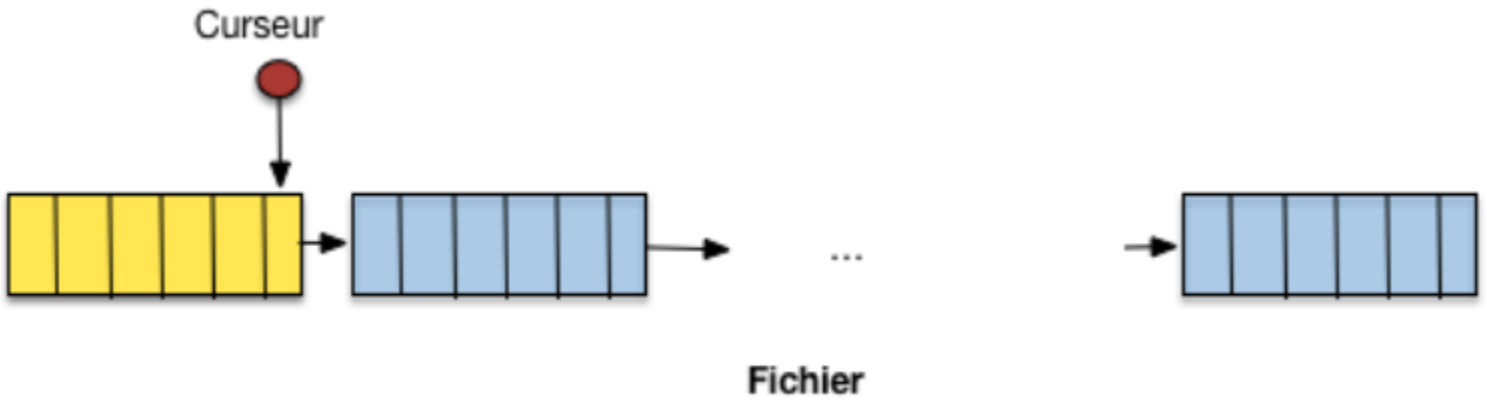
Le premier next() entraîne l'accès au premier bloc, placé en mémoire.



Le curseur se place sur le premier tuple, qui est retourné comme résultat.
Le temps de réponse est minimal.

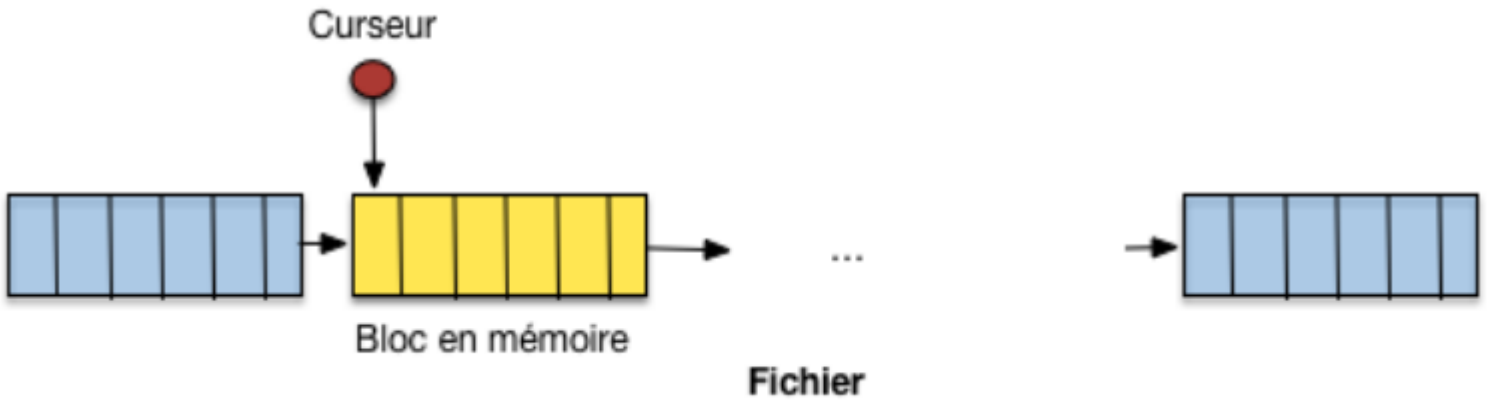
Parcours Séquentiel : FullScan

Après plusieurs next(), le curseur est positionné sur le dernier tuple du bloc.



Parcours Séquentiel : FullScan

L'appel suivant à next() charge le second bloc en mémoire

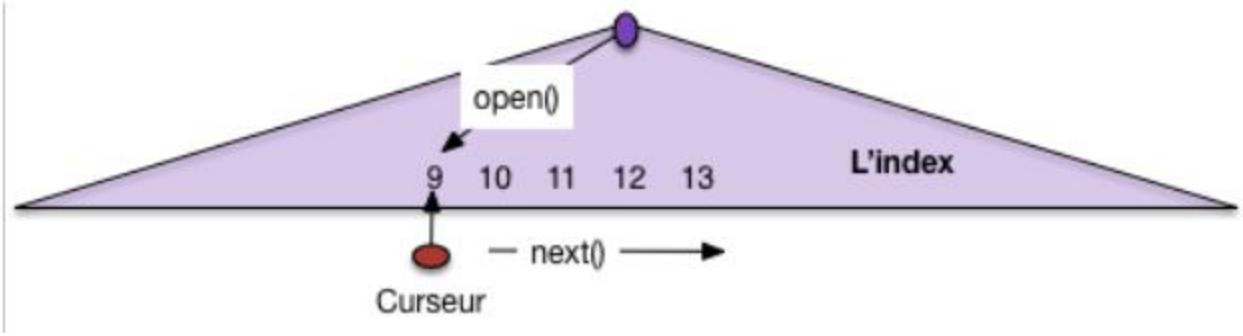


Bilan :

- besoin en mémoire réduit (1 bloc) ;
- temps de réponse très court.

Parcours par Index : IndexScan

- Pendant la phase **open()** : Il y'a un parcours de la racine vers la feuille.
- A chaque appel à next() : parcours en séquence des feuilles.

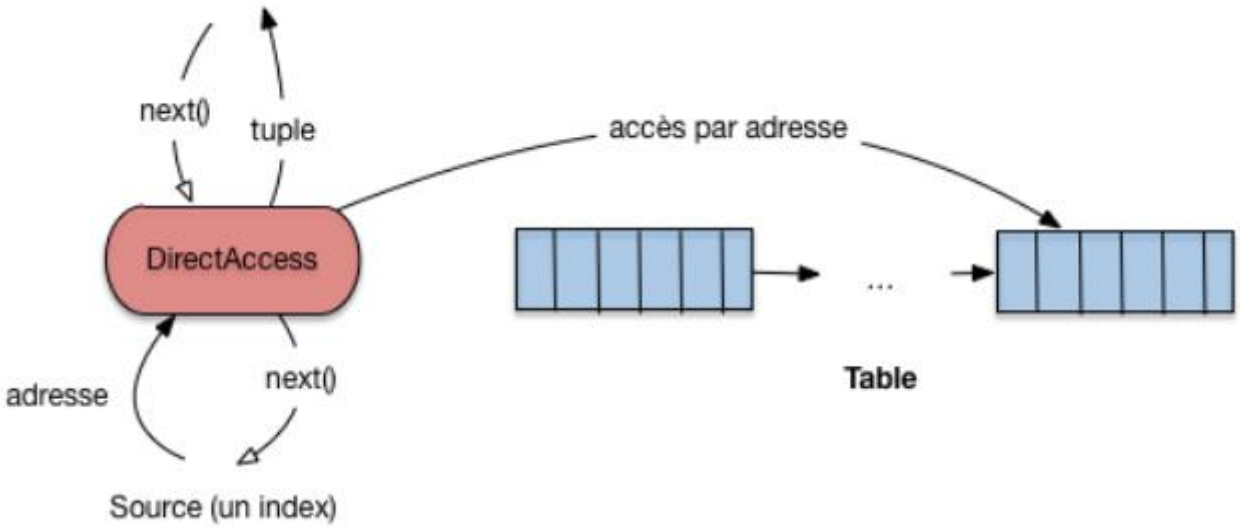


Bilan :

- très efficace, quelques lectures logiques (l'index est généralement dans la mémoire tampon)

Accès par Adresse: DirectAccess

- Pendant le open() : rien à faire.
- A chaque appel à next() : on reçoit une adresse, on produit un tuple

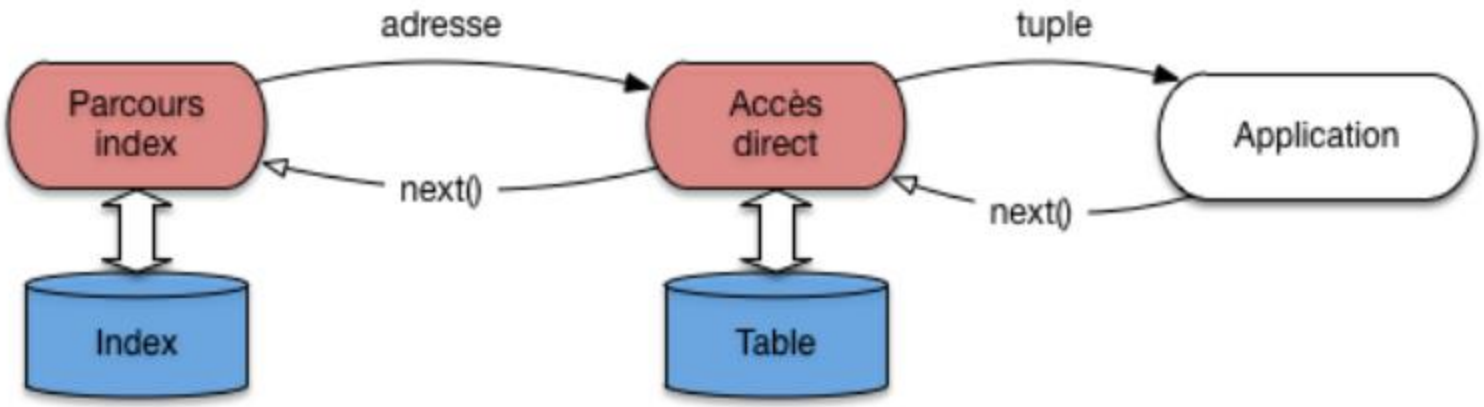


Bilan :

- Très efficace : un accès bloc, souvent dans la mémoire tampon.

Plan d'exécution

- Un plan d'exécution connecte les opérateurs.



Exemple de plans d'exécution

Nous allons étudier les plans permettant d'exécuter les requêtes portant sur une seule table.

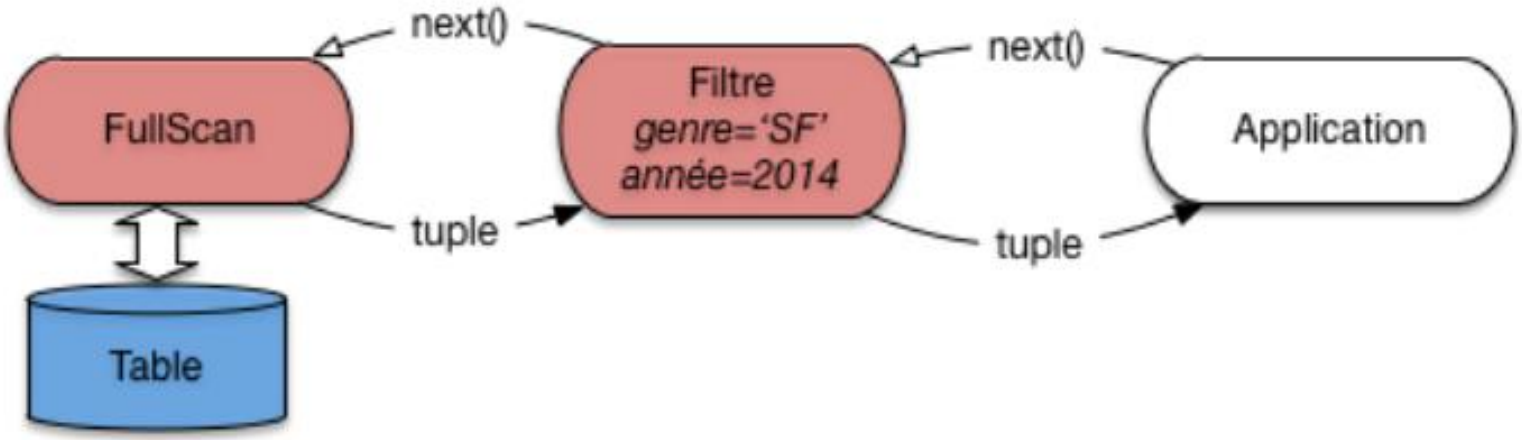
```
select a1, a2, ..., an
from T
where condition
```

- Quels opérateurs doit-on utiliser ?
  - [FullScan] : parcours séquentiel de la table.
  - [IndexScan] : parcours d'un index (si disponible).
  - [DirectAccess] : accès par adresse à un tuple.
  - [Filter] : test de la condition.

Optimisation Physique

Plan 1 : Sans Index

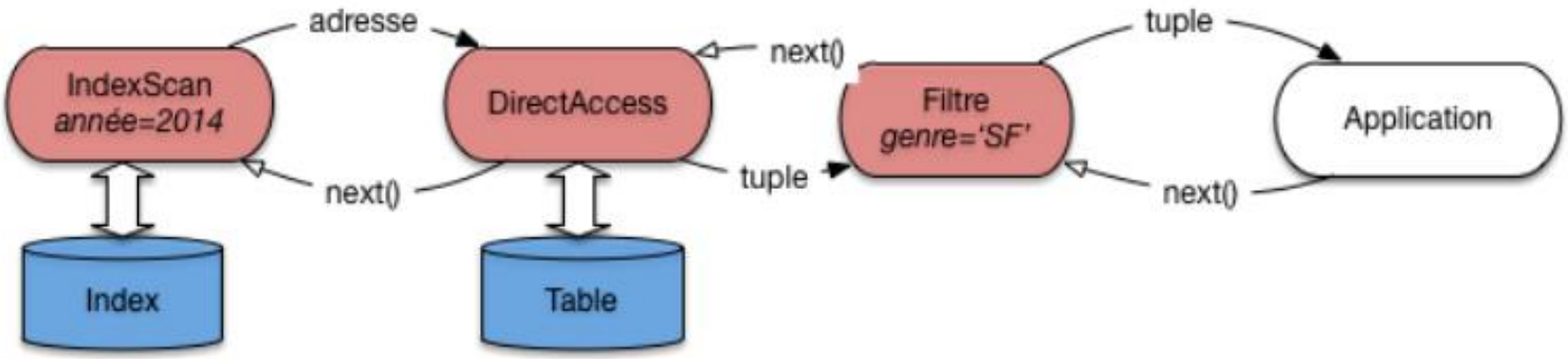
```
select titre from Film where genre='SF' and annee = 2014
```



Optimisation Physique

Plan 2 : Avec Index sur Année

```
select titre from Film where genre='SF' and annee = 2014
```



Algorithmes de Jointure

Algorithmes de jointure de deux relations R, S

- Boucles imbriquées :
  - sans index
  - Avec index : index B+
- Tri-fusion
- Jointure **par hachage**

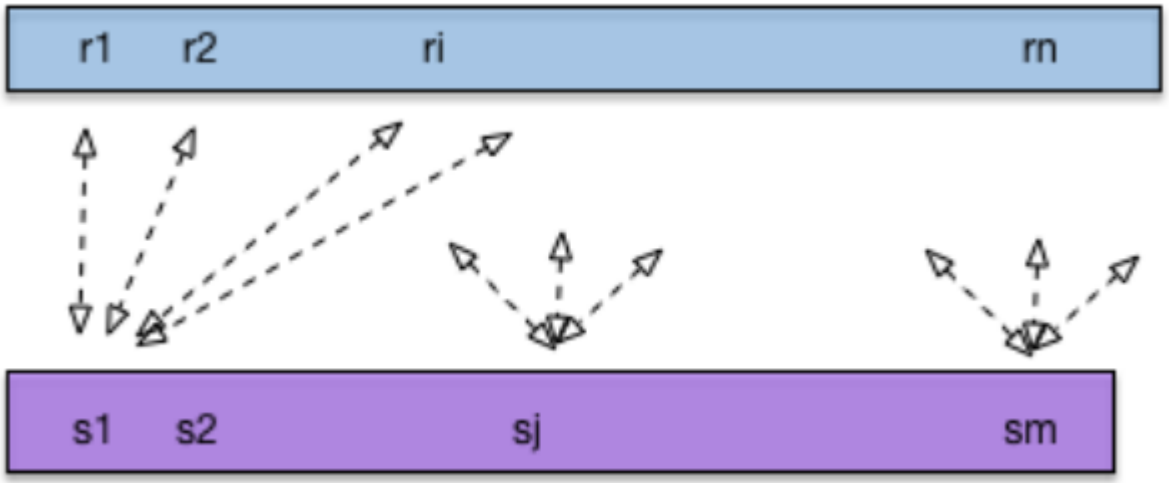
Jointure par boucle imbriquées (sans index)

Principe : Jointure (R, S)

compare chaque tuple de la relation R à tous les tuples de la relation S,

- concaténer les tuples pour lesquels la condition de jointure est vérifiée
- ajouter le nouveau tuple au résultat.

Idée : Evaluation tuple par tuple



Jointure par boucle imbriquées (sans index)

Principe : Jointure (R, S) iterative brute

compare chaque tuple de la relation R à tous les tuples de la relation S,

- concaténer les tuples pour lesquels la condition de jointure est vérifiée
- ajouter le nouveau tuple au résultat.

Idée : Evaluation tuple par tuple

```
T = {};
Pour chaque tuple r de R faire
    Pour chaque tuple s de S faire
      Si r.A==s.A Alors T = T ∪ (r.A, r.B, s.C)
    FinPour
FinPour
```

Jointure par boucle imbriquées (sans index)

Coût de la Jointure (R, S)

- On parcourt la relation R autant de fois que R a de pages : **M opérations d'E/S**
- On parcourt S autant de fois que R a de tuples (**pR*M**) et à chaque passage on est obligé de charger toutes les pages de S (N)
- Au total : **M + pR*M*N**

Remarque

R et S sont deux tables

- **R : M pages,**
- **pR : est le nombre d'enregistrements/page pour la table R**
- **S : N pages,**
- **pS : est le nombre d'enregistrements/page pour la table S**

Algorithmes de Jointure

Jointure avec Index

Idèe :

On exploite la structure **en pages** de R et S.

Pour chaque page de R, on peut retrouver chaque page de S et écrire les tuples <r,s> satistfaisant la condition de jointure

Algorithme :

Pour chaque **page de R** faire

Pour chaque **page de S** faire

si r.a == s.a alors ajouter <r, s> au résultat

- On gagne un facteur de **pR,** c-à-d., le coût est divisé par pR

Le coût = M + M* N

Optimisation: on choisit la relation la plus petite en nombre de pages dans la boucle extérieure (telle que M < N)

Jointure par Tri-Fusion

Cet algorithme est utile quand il n'y a pas d'index sur des tables de grandes tailles

Principe :

- Pour chaque tuple de **R** on utilise l'index pour retrouver les tuples correspondant de **S** (c'est S qui doit être indéxée).
- On compare **r** seulement avec les tuples de **S** qui ont la même valeur sur la colonne de jointure

Jointure par Tri-Fusion

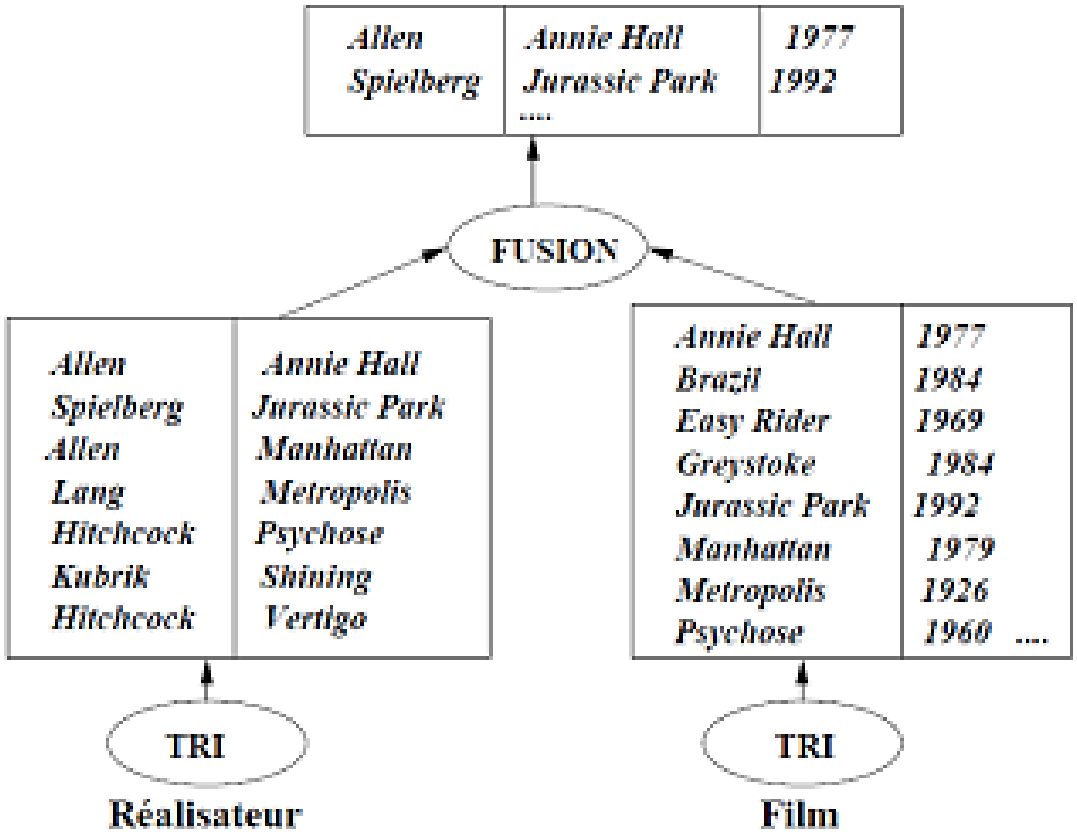
Algorithme utile quand il n'y a pas d'index sur des tables de grandes tailles

Principe : jointure(R, S)

- Trier R et S par rapport à l'attribut de jointure A
- Fusionner R et S :
  - Positionner un pointeur courant **ptR** et **ptS** respectivement sur le premier tuple de **R** et **S**
  - Comparer les attributs de jointure de ces deux tuples.
  - Si l'attribut **A** du tuple pointé par **ptR** est égal à l'attribut **A** du tuple pointé par **ptS**
    - Alors on génère un tuple résultat et on incrémente **ptR**
  - Sinon, si l'attribut **A** du tuple pointé par **ptR** est supérieur à l'attribut **A** du tuple pointé par **ptS**,
    - Alors on incrémente **ptS**
  - Sinon, si l'attribut A du tuple pointé par **ptR** est inférieur à l'attribut A du tuple pointé par **ptS**
    - Alors on incrémente **ptR** afin qu'il pointe sur le tuple suivant de **R**.

Jointure par Tri-Fusion

Réalisateur (nom, titre) ▷◁ ***Film*** (titre, année)



Algorithmes de Jointure

Jointure avec Index

Idèe :

Pour chaque tuple de R, on utilise l'index pour retrouver les tuples correspondant de S (La table S doit être indéxèe). On compare r seulement avec les tuples de S qui ont la même valeur sur la colonne de jointure

⟹ **On n'énumère donc pas le produit cartésien**

Algorithme :

Pour **chaque tuple de R** faire
    Pour **chaque tuple de S** où r.a == s.a
        faire ajouter <r, s> au résultat

- Le coût pour scanner R est toujours M
- Le coût pour retrouver les tuples correspondant de S dépen du type d'index et du nombre de tuples qui correspondent à l'attribut de jointure

Le coût = M * pR * M* coût d'accès index + coût index →données

Algorithmes de Jointure

Exemple de jointure avec Index

- Les films et leur metteur en scène

```
select * from Film as f, Artiste as a
where f.id_realisateur = a.id
```

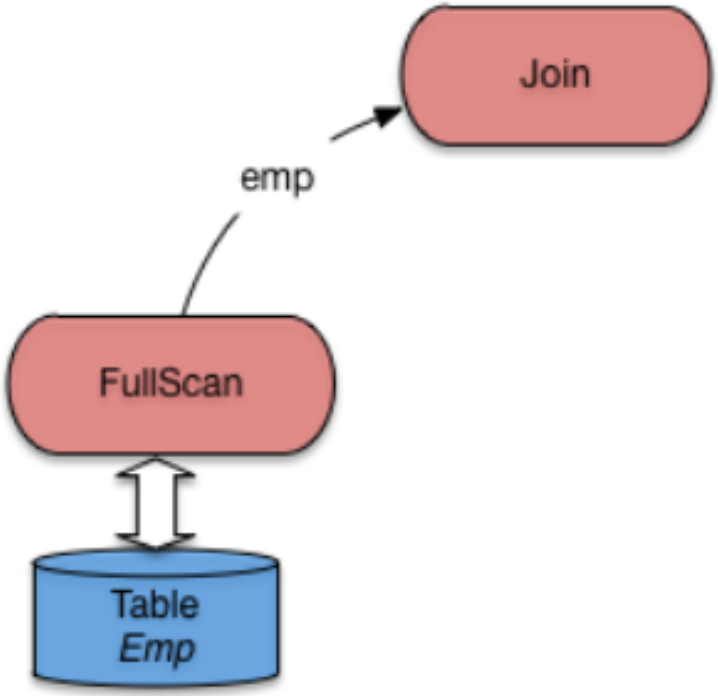
- Les employés et leur département

```
select * from emp e, dept d
where e.dnum = d.num
```

Algorithmes de Jointure

Exemple de jointure avec Index

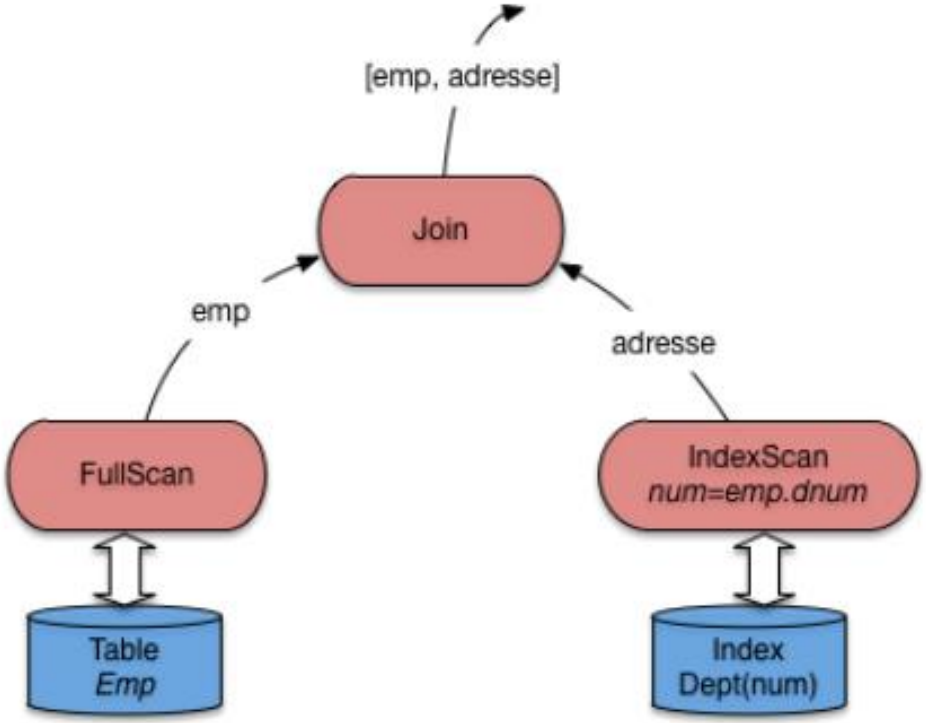
On parcourt séquentiellement la table contenant la clé étrangère.



On obtient des nuplets `employé`, avec leur no de département.

Exemple de jointure avec Index

On utilise le no de département pour un accès à l'index Dept.

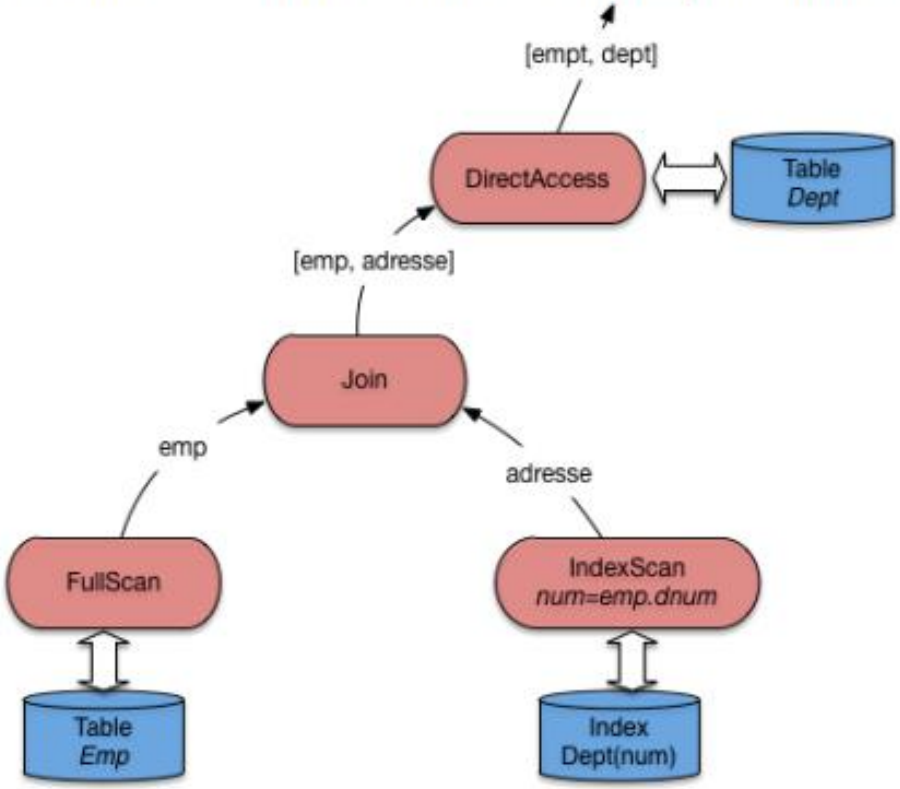


On obtient des paires [employé, adrDept].

Algorithmes de Jointure

Exemple de jointure avec Index

Il reste à résoudre l'adresse du département avec un accès direct.



On obtient des paires [employé, dept].

Ressources utilisées

- Cours de bases de données, aspects systèmes, http://sys.bdpedia.fr
- Bases de données avancées : Optimisation des opérateurs.
  Cours réalisé par Sarah Cohen-Boulakia