

SGBD-R : traitement de requêtes

L'optimiseur

Guillaume Raschia — Nantes Université

Dernière mise-à-jour : 20 janvier 2023

originaux de Philippe Rigaux, CNAM

Plan de la session

L'évaluation d'une requête (S7.1)

De SQL au PEL (S7.2)

L'optimisation (S7.3)

Avec ORACLE (S7.4)

L'évaluation d'une requête (S7.1)

Le problème étudié



Une requête SQL est **déclarative**. Elle ne dit pas **comment** calculer le résultat.

Nous avons besoin d'une **forme opératoire** : un programme.

La notion de plan d'exécution

```
select a1, a2, ...  
from T1, T2, ...  
where ...
```

Forme
déclarative

?



Forme
opérateur

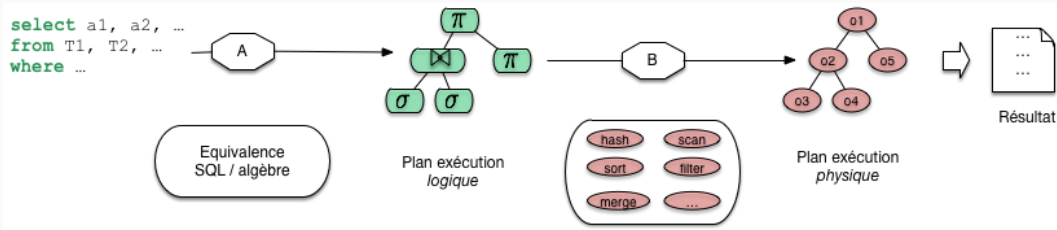


Résultat

Dans un SGBD le programme qui exécute une requête est appelé **plan d'exécution**.

Il a une forme particulière : c'est un **arbre**, constitué d'**opérateurs**.

De la requête SQL au plan d'exécution



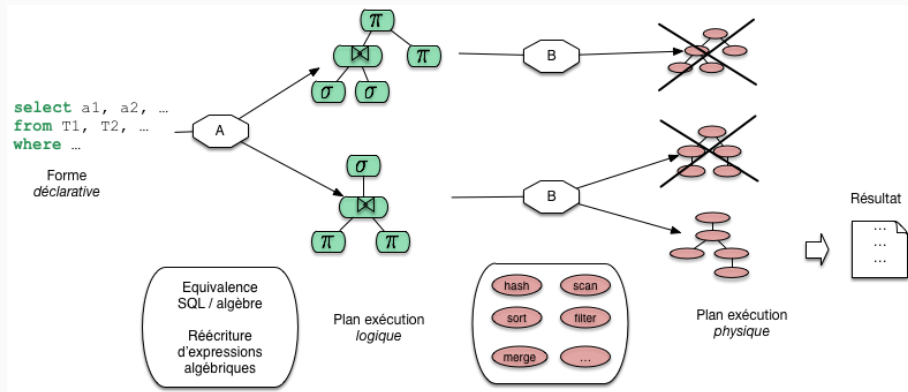
Deux étapes :

(A) plan d'exécution **logique** (expression algébrique);

(B) plan d'exécution **physique** (arbre d'itérateurs).

Le SGBD s'appuie sur un **catalogue** d'opérateurs.

En quoi consiste l'optimisation ?



À chaque étape, plusieurs choix. Le système les évalue et choisit le « meilleur ».

Étapes du traitement d'une requête

Toute requête SQL est traitée en trois étapes :


1. **Analyse et traduction** de la requête. On vérifie qu'elle est correcte, et on l'exprime sous forme d'opérations.
2. **Optimisation** : comment agencer au mieux les opérations, et quels algorithmes utiliser. On obtient un **plan d'exécution**.
3. **Exécution de la requête** : le plan d'exécution est compilé et exécuté.

Ingrédients du traitement d'une requête

Le traitement s'appuie sur les éléments suivants :

- Le schéma de la base, la description des tables
- L'organisation physique des données (chemins d'accès)
- Les statistiques : taille des tables, des index, distribution des valeurs¹
- Un catalogue d'opérateurs : il peut différer d'un système à l'autre

Important : on suppose que le temps d'accès à ces informations est négligeable par rapport à l'exécution de la requête.

1.  Taille du résultat intermédiaire $\sigma_{A=a}(R)$? avec histogramme ? avec la taille du domaine actif ?

Un SGBD

- construit un **plan d'exécution** en suivant une **chaîne d'évaluation** complexe,
- applique ce plan pour produire le résultat d'une requête,
- choisit ce plan en fonction de **critères d'optimisation**.

De SQL au PEL (S7.2)

Les blocs SQL

Une requête SQL est décomposée en blocs, et **une optimisation s'applique à chaque bloc**.

Un bloc est une requête **select-from-where** sans imbrication.

Exemple

```
|      select titre  
|      from    Film  
|      where   année = (select min (année) from Film)
```

Premier bloc : calcule une valeur v .

```
|      select min (année) from Film
```

Second bloc : utilise v comme critère.

```
|      select titre from Film where année =  $v$ 
```

Quelle est l'influence du découpage en blocs ?

En principe, le système devrait pouvoir déterminer les requêtes équivalentes, indépendamment de la syntaxe.

En réalité, ça ne se passe pas tout à fait comme ça...

Dans quel film paru en 1958 joue James Stewart ?

Expression SQL « à plat » :

```
select titre
from   Film f, Rôle r, Artiste a
where  a.nom = 'Stewart' and a.prénom='James'
and    f.id = r.id_film
and    r.id_acteur = a.id
and    f.année = 1958
```

Une autre syntaxe pour la même requête

Expression SQL équivalente, avec `in`.

```
select titre
from Film f, Rôle r
where f.id = r.id_film
and f.année = 1958
and r.id_acteur in (select id
                    from Artiste a
                    where a.nom = 'Stewart'
                    and a.prénom = 'James')
```

C'est plus clair comme ça ?

```
select titre from Film f
where f.année = 1958
and    f.id in
      (select r.id_film from Rôle r
       where r.id_acteur in
            (select a.id
             from Artiste a
             where a.nom = 'Stewart'
             and a.prénom = 'James'))
```

Encore une autre syntaxe

Expression SQL équivalente, avec `exists`.

```
select titre
from   Film f, Rôle r
where  f.id = r.id_film
and    f.année = 1958
and    exists (select 1
               from Artiste a
               where nom = 'Stewart'
               and prénom = 'James'
               and r.id_acteur = a.id)
```

Requête imbriquée corrélée (`r.id_acteur=a.id`)
évaluée pour chaque `r.id_acteur`!

Une dernière

Maintenant, avec deux quantificateurs `exists`.

```
select titre from Film f
where année = 1958
and exists
    (select 1 from Rôle r
     where r.id_film = f.id
     and exists
         (select 1
          from Artiste a
          where a.id = r.id_acteur
          and nom = 'Stewart'
          and prénom = 'James'))
```

Important : On risque de figer la manière dont le système évalue la requête.

Pourquoi est-ce mauvais ?

Par exemple, la dernière version aboutit à un plan d'exécution sous-optimal.

1. On parcourt tous les films parus en 1958
2. Pour chaque film : on cherche les rôles du film
3. Sans index, on doit parcourir tous les rôles, **pour chaque film**
4. **Pour chaque rôle**, on regarde si c'est James Stewart

Ça va coûter cher!!

Pourquoi n'y a-t'il pas d'index disponible sur Rôle ?

La table Rôle, avec une clé composite.

```
create table Rôle (id_acteur integer not null,  
                  id_film integer not null,  
                  nom_rôle varchar(30) not null,  
                  primary key (id_acteur, id_film),  
                  foreign key (id_acteur) references Artiste(id),  
                  foreign key (id_film) references Film(id),  
                  );
```

Regardez bien la clé primaire, et pensez à l'arbre B+ associé. Peut-on l'utiliser ?

- Recherche sur `id_acteur` et `id_film`. Oui.
- Recherche sur `id_acteur`. Oui.
- Recherche sur `id_film`. Non.

Conclusion : requêtes SQL et blocs

La leçon : mieux vaut écrire les requêtes SQL « à plat », en un seul bloc, et laisser le système décider du meilleur agencement des accès.

Confronté à des requêtes SQL à plusieurs blocs : Étudier le plan d'exécution pour vérifier que les index sont correctement utilisés.

On va voir comment faire par la suite.

Traitement d'un bloc

Plusieurs phases :

1. **Analyse syntaxique** : conformité SQL, conformité au schéma.
2. **Transformations SQL** : mise à plat des requêtes imbriquées, simplifications logiques, etc.
3. **Vérification de cohérence** : détection de clauses insatisfiables telles que `année < 2000 and année > 2001`
4. **Traduction** en une expression algébrique, plus « opérationnelle » que SQL.

En résumé

Toute requête SQL bien formée se traduit en une expression de l'**algèbre relationnelle étendue** (Pensons à `group by`, `having`, `order by`...).

Exemple de traduction

La requête : « le titre du film paru en 1958, où l'un des acteurs joue le rôle de John Ferguson »
(rép. : *Vertigo*, traduit par *Sueurs Froides*)

En SQL :

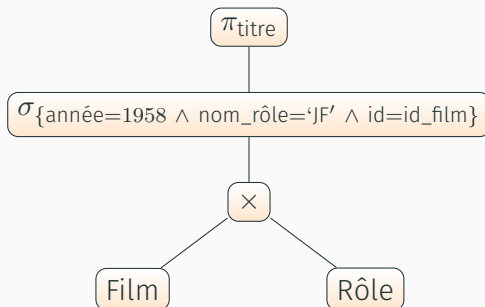
```
select titre
from Film f, Rôle r
where nom_rôle = 'John Ferguson'
and f.id = r.id_film
and f.année = 1958
```

En algèbre :

$$\pi_{\text{titre}} \left(\sigma_{\{\text{année}=1958 \wedge \text{nom_rôle}='JF' \wedge \text{id}=\text{id_film}\}} (\text{Film} \times \text{Rôle}) \right)$$

Le Plan d'Exécution Logique (PEL)

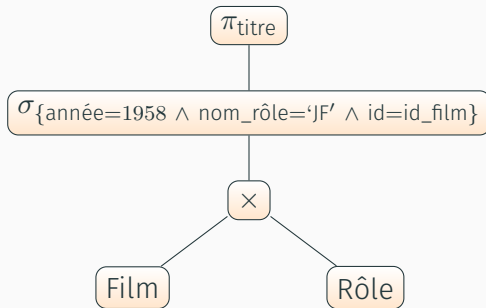
L'expression algébrique canonique nous donne une ébauche de plan d'exécution.



Est-ce le bon? Pas forcément : **l'optimisation** va nous permettre d'en trouver d'autres et de les comparer.

L'optimisation (S7.3)

« le titre du film paru en 1958, où l'un des acteurs joue le rôle de John Ferguson »



Trois sélections (l'année, le rôle, le film), un produit cartésien et une projection.

On engage alors une phase de **réécriture** pour en produire une version **a priori** plus efficace.

Il y a plusieurs expressions **équivalentes** pour une même requête.

On peut explorer ces expressions grâce à des règles de réécriture.

Exemple

- **Décomposition des sélections** : $\sigma_{A=a \wedge B=b}(R) \equiv \sigma_{A=a}(\sigma_{B=b}(R))$
- **Commutativité de σ et de π** : $\pi_{A_1, A_2, \dots, A_p}(\sigma_{A_i=a}(R)) \equiv \sigma_{A_i=a}(\pi_{A_1, A_2, \dots, A_p}(R))$
- **Commutativité de σ et de \bowtie** : $\sigma_{A=a}(R[\dots A \dots] \bowtie S) \equiv \sigma_{A=a}(R) \bowtie S$

- Formation des jointures : $\sigma_{A=B}(R[\dots A \dots] \times S[\dots B \dots]) \equiv R \bowtie_{A=B} S$
- Commutativité des jointures : $R \bowtie S \equiv S \bowtie R$
- Associativité des jointures : $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$
- etc.

L'application dirigée d'une règle de réécriture transforme une expression e en une expression e' **équivalente**.

Ce que fait l'optimiseur

Trouve les expressions équivalentes, évalue leur coût et choisit la meilleure.

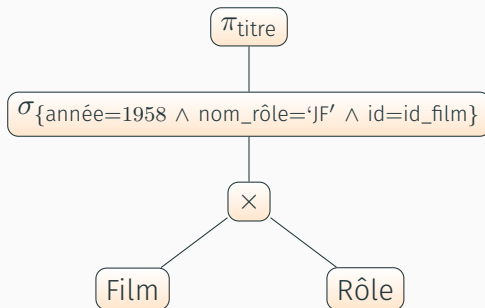
Important : on ne peut pas énumérer tous les plans possibles (trop long) : on applique des heuristiques.

Heuristique classique : réduire la taille des données

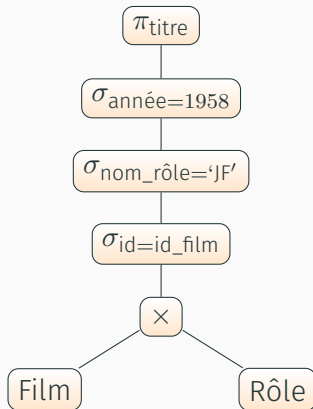
- en filtrant les nuplets par des sélections
- en les simplifiant par des projections

dès que possible.

Reprenons : le film paru en 1958 avec un rôle 'Ferguson'.

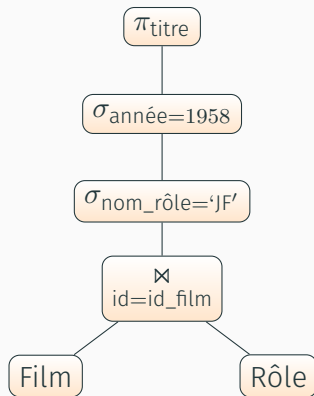


Première étape : je sépare les sélections.

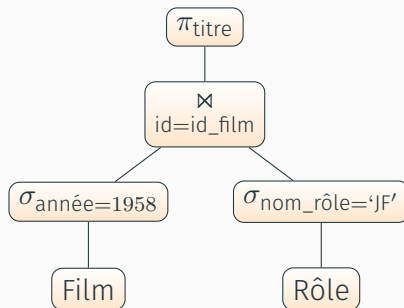


Illustration

Ensuite je construis la jointure par composition de la sélection avec le produit cartésien.



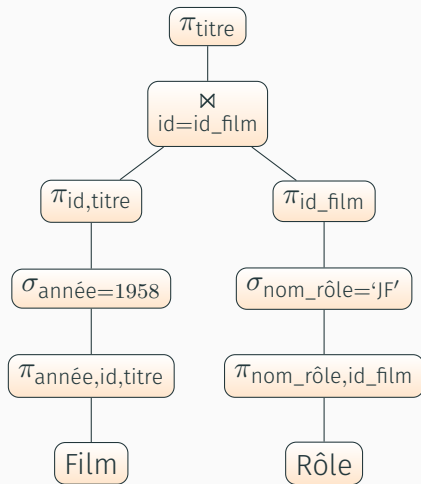
Puis je « pousse » chaque sélection vers sa table.



✎ Dans quelles circonstances est-il préférable d'évaluer la jointure avant la sélection ?

Illustration

Et enfin, je « pousse » également les projections vers les tables.



Principes essentiels :

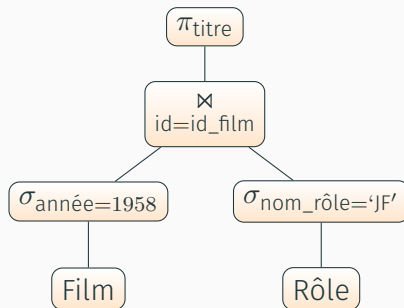
- L'algèbre permet d'obtenir une version opératoire de la requête.
- Les équivalences algébriques produisent un ensemble de PELs.
- L'optimiseur évalue le coût de chaque plan.

Bien retenir

- **Heuristique** : on ne peut pas **tout explorer**.
- **Étape nécessaire mais pas suffisante** : il reste à choisir le bon algorithme pour chaque opération.

L'ordre des jointures multiples est étudié au passage vers le plan d'exécution physique (PEP).

Notre requête sur les films avec John Ferguson, après optimisation algébrique, et en omettant les projections intermédiaires.



Il nous restait à choisir les chemins d'accès et les algorithmes de jointure.

Continuons à optimiser notre requête

Hypothèse : le système cherche à utiliser les index pour les jointures.

$$\text{Film} \bowtie_{\text{id=id_film}} \text{Rôle}$$

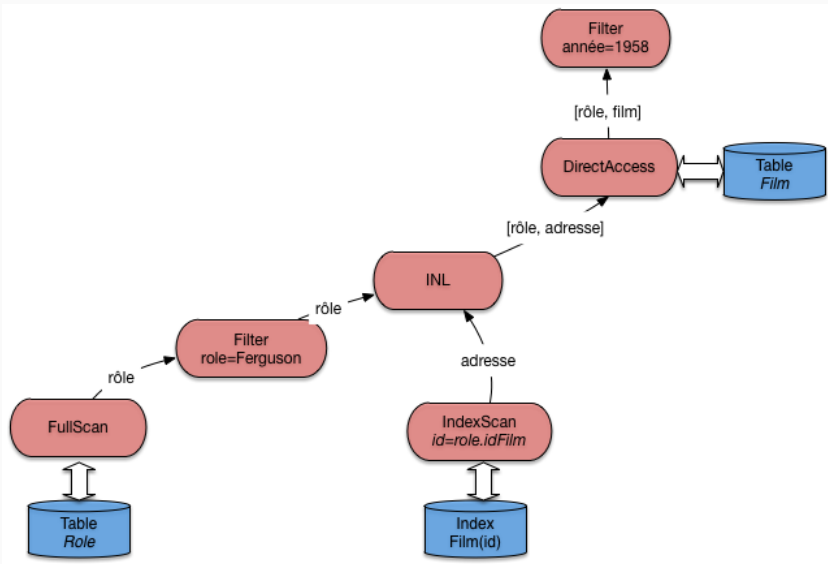
ou

$$\text{Rôle} \bowtie_{\text{id_film=id}} \text{Film}$$

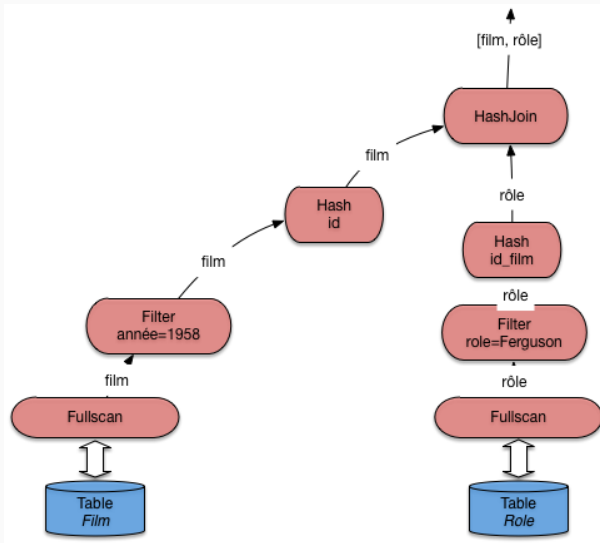
Après analyse des index, la bonne forme est

$$\pi_{\text{titre}} \left(\sigma_{\text{année}=1958} \left(\sigma_{\text{nom_rôle}='JF'}(\text{Rôle}) \bowtie_{\text{id_film=id}} \text{Film} \right) \right)$$

Le plan d'exécution



Et si je n'ai pas d'index?



Les jointures multiples

```
select *  
from Film, Rôle, Artiste, Pays  
where Pays.nom='Islande'  
and Film.id=Rôle.id_film  
and Rôle.id_acteur=Artiste.id  
and Artiste.pays = Pays.code
```

Film ⋈ Rôle ⋈ Artiste ⋈ Pays

Le cœur de la requête est une chaîne de jointures naturelles.

Les jointures multiples

Film ⋈ Rôle ⋈ Artiste ⋈ Pays

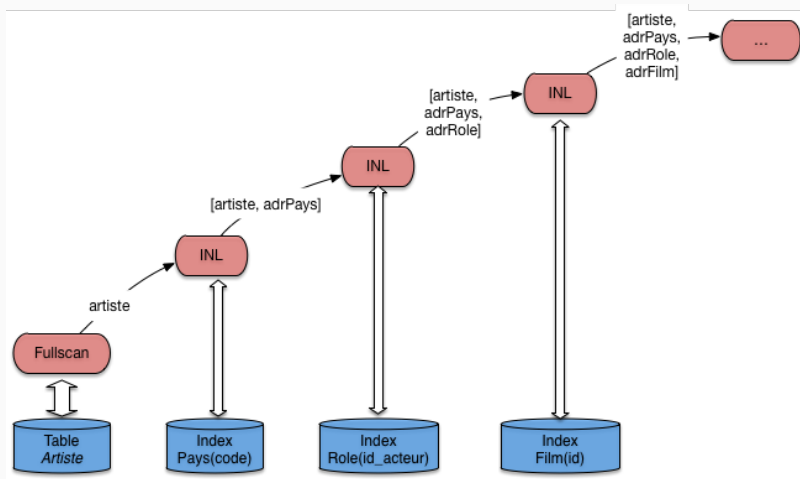
Se résoud successivement avec l'algorithme de **boucle imbriquée indexée (INL)**.

Les index sur clés primaires guident l'ordre des jointure.

Un bon ordre

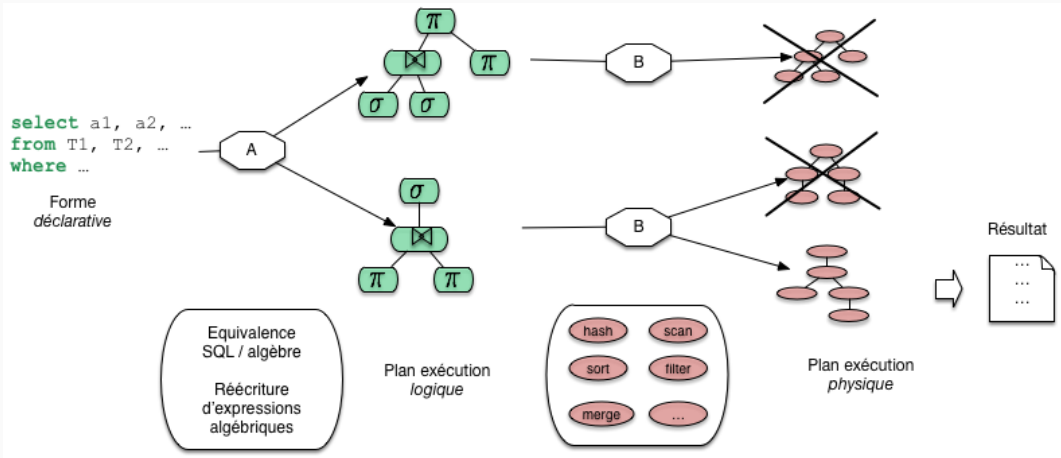
((Artiste ⋈ Pays) ⋈ Rôle) ⋈ Film

Les jointures multiples



Arbre en profondeur à gauche, ou plan « en peigne ».

Pour mémoire : la phase d'optimisation



Avec ORACLE (S7.4)

L'optimiseur ORACLE suit une approche classique :

1. Génération de plusieurs plans d'exécution.
2. Estimation du coût de chaque plan généré.
3. Choix du meilleur et exécution.

Tout ceci est automatique, mais il est possible d'**influer**, voire de **forcer** le plan d'exécution.

Ceux que nous avons déjà vus.

- **Parcours séquentiel** : FULL TABLE SCAN.
- **Par adresse** : TABLE ACCESS BY INDEX ROWID.
- **Parcours d'index** : INDEX SCAN
décliné en plusieurs versions (unique, range, full, etc.).

Opérations physiques

Voici les principales :

- **FILTER** : élimination de n-uplets (sélection).
- **INTERSECTION** : intersection de deux ensembles de n-uplet.
- **UNION** : union de deux ensembles.
- **MINUS** : différence de deux ensembles.

Les projections sont systématiquement combinées aux opérations, et consultables à la rubrique **Column Projection Info**.

D'autres opérations sont liées aux algorithmes de jointures, à l'agrégation et à de multiples extensions SQL.

ORACLE utilise trois algorithmes de jointure :

- **Boucles imbriquées** quand il y a au moins un index.
 - Opération **NESTED LOOPS**.
- **Tri-fusion** quand il n'y a pas d'index.
 - Opérations **SORT** et **MERGE**.
- **Jointure par hachage** quand il n'y a pas d'index.
 - Opération **HASH JOIN**.

L'outil EXPLAIN PLAN

L'outil EXPLAIN PLAN donne le plan d'exécution d'une requête.

Dans sa version simple, la description comprend :

- Le chemin d'accès utilisé.
- Les opérations physiques (tri, fusion, intersection, ...).
- L'ordre des opérations.

Il est représentable par un arbre.

Quelques exemples

Rappel du schéma

- Film (**idFilm**, titre, année, genre, résumé, *idRéalisateur*, *codePays*)
- Artiste (**idArtiste**, nom, prénom, annéeNaissance)
- Rôle (*idActeur*, *idFilm*, nomRôle)
- Internaute (**email**, nom, prénom, région)
- Notation (*email*, *idFilm*, note)
- Pays (**code**, nom, langue)

Sélection sans index

La requête :

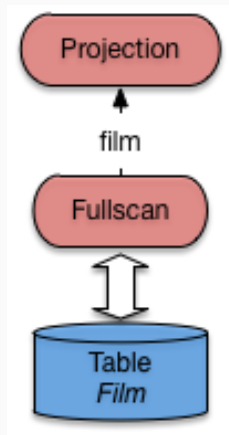
```
explain plan
set statement_id='SelSansInd' for
select *
from Film
where titre = 'Vertigo'
```

Le résultat de EXPLAIN PLAN :

```
0 SELECT STATEMENT
1 TABLE ACCESS FULL FILM
```

Plan d'exécution

On ne fait pas plus simple !



Sélection avec index

La requête :

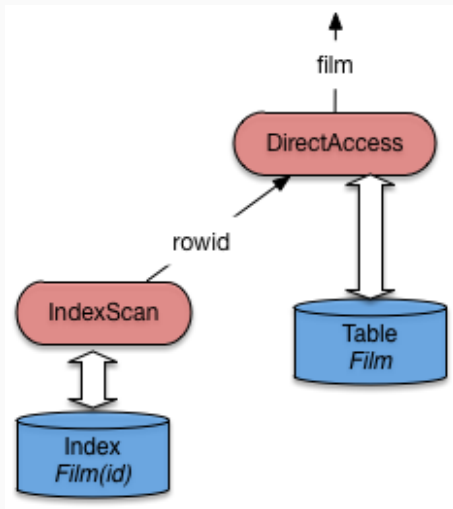
```
explain plan  
set statement_id='SelInd' for  
select *  
from Film  
where id=21;
```

Le résultat de EXPLAIN PLAN :

```
0 SELECT STATEMENT  
1 TABLE ACCESS BY INDEX ROWID FILM  
2 INDEX UNIQUE SCAN IDX-FILM-ID
```

Plan d'exécution

Accès à l'index, puis à la table.



Jointure avec index

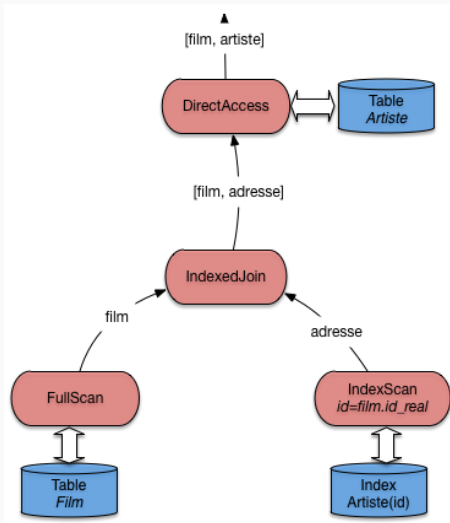
La requête :

```
explain plan
set statement_id='JoinIndex' for
select titre, nom, prénom
from   Film f, Artiste a
where  f.id_réalisateur = a.id;
```

Le résultat de EXPLAIN PLAN :

```
0  SELECT STATEMENT
   1  NESTED LOOPS
      2  TABLE ACCESS FULL FILM
      3  TABLE ACCESS BY INDEX ROWID ARTISTE
          4  INDEX UNIQUE SCAN IDX-ARTISTE-ID
```

Plan d'exécution (déjà vu)



Jointure avec index et sélection

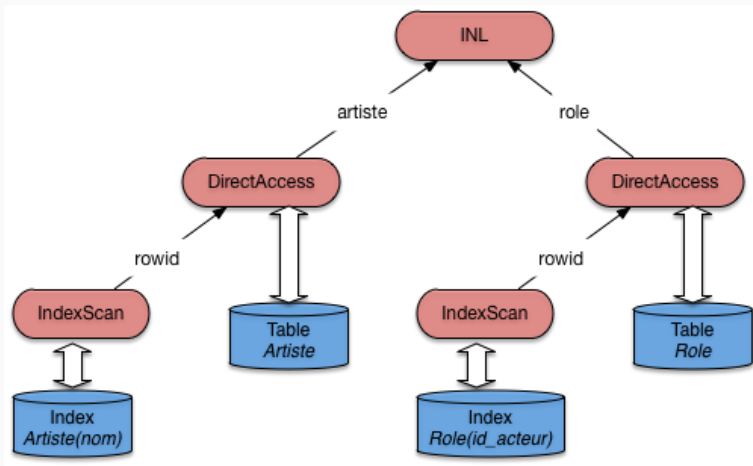
La requête (avec index sur le nom des artistes) :

```
explain plan
set statement_id='JoinSelIndex' for
select nom_rôle
from   Rôle r, Artiste a
where  r.id_acteur = a.id
and    nom = 'Pacino';
```

Le résultat de EXPLAIN PLAN :

```
0 SELECT STATEMENT
1 NESTED LOOPS
2 TABLE ACCESS BY INDEX ROWID ARTISTE
3 INDEX RANGE SCAN IDX-ARTISTE-NOM
4 TABLE ACCESS BY INDEX ROWID RÔLE
5 INDEX RANGE SCAN IDX-RÔLE-ID-ACTEUR
```


Plan d'exécution



Jointure sans index

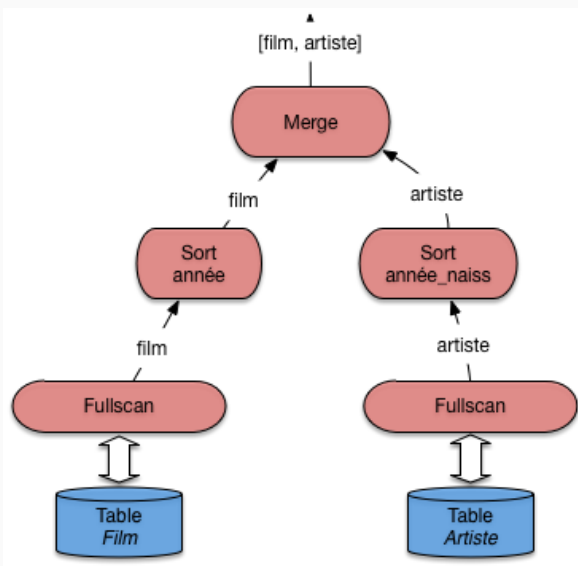
Requête :

```
explain plan set
statement_id='JoinSansIndex' for
select nom, prenom
from   Film f, Artiste a
where  f.année  = a.année_naissance
and    titre = 'Vertigo';
```

Le résultat de EXPLAIN :

```
0  SELECT STATEMENT
   1  MERGE JOIN
     2  SORT JOIN
       3  TABLE ACCESS FULL ARTISTE
     4  SORT JOIN
       5  TABLE ACCESS FULL FILM
```

Plan d'exécution



Ce chapitre explique comment, pour produire un résultat de requête, un SGBD

- construit et applique un **plan d'exécution**
- choisit ce plan en fonction de **critères d'optimisation**.

Ce que vous avez acquis à l'issue du chapitre :

- Compréhension de la manière dont une requête est exécutée.
- Capacité à interpréter le plan d'exécution (outil **explain**).
- Rudiments pour explorer et ajuster les paramètres « physiques » qui conditionnent les performances d'un système.

Important

Sujet complexe : le cours présente les bases, à approfondir.