

Cours de bases de données,  
aspects systèmes,  
<http://sys.bdpedia.fr>

Traitement d'une requête

# Étapes du traitement d'une requête

Toute requête SQL est traitée en trois étapes :

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

# Support du traitement d'une requête

Le traitement s'appuie sur les éléments suivants :

- **Le schéma de la base**, description des tables et chemins d'accès (dans le catalogue)
- **Des statistiques** : taille des tables, des index, distribution des valeurs
- **Des opérateurs** : il peuvent différer selon les systèmes

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

# Les blocs SQL

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

Un bloc est une requête “select-from-where” sans imbrication. Exemple :

```
select titre
from Film
where annee = (select min (annee) from Film)
```

Premier bloc : calcule une valeur  $v$ .

```
select min (annee) from Film
```

Second bloc : utilise  $v$  comme critère.

```
select titre from Film where annee = v
```

# Quelle est l'influence du découpage en blocs ?

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

En principe, 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, Role r, Artiste a
where  a.nom = 'Stewart' and a.prenom='James'
and    f.id_film = r.id_film
and    r.id_acteur = a.idArtiste
and    f.annee = 1958
```

# Une autre syntaxe pour la même requête

Expression SQL équivalente, avec in.

```
select titre
from Film f, Role r
where f.id_film = r.id_film
and f.annee = 1958
and r.id_acteur in (select id_acteur
                    from Artiste
                    where nom='Stewart'
                    and prenom='James')
```

# Encore une autre syntaxe

Expression SQL équivalente, avec exists.

```
select titre
from Film f, Role r
where f.id_film = r.id_film
and f.annee = 1958
and exists (select 'x'
            from Artiste a
            where nom='Stewart'
            and prenom='James'
            and r.id_acteur = a.id_acteur)
```

# Avec encore plus de blocs

C'est plus clair comme ça ?

```
select titre from Film
where annee = 1958
and id_film in
    (select id_film from Role
     where id_acteur in
         (select id_acteur
          from Artiste
          where nom='Stewart'
          and prenom='James'))
```

# Une dernière

Maintenant, avec exists.

```
select titre from Film
where annee = 1958
and exists
  (select * from Role
   where id_film = Film.id
   and exists
     (select *
      from Artiste
      where id = Role.id_acteur
      and nom='Stewart'
      and prenom='James'))
```

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

# Pourquoi c'est mauvais ?

Les deux dernières versions aboutissent à un plan d'exécution sous-optimal.

On parcourt tous les films parus en 1958

Pour chaque film : on cherche les rôles du film, **mais pas d'index disponible**

Donc pas d'autre solution que de parcourir tous les rôles, **pour chaque film.**

Ensuite, pour chaque rôle on regarde si c'est James Stewart

Ca va coûter cher !!

# Comprendre : pourquoi pas d'index disponible sur Role ?

La table Role, avec une clé composite.

```
create table Role (id_acteur integer not null,  
                  id_film integer not null,  
                  nom_role 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 : **Etudier 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 :

**Analyse syntaxique** : conformité SQL, conformité au schéma.

**Analyse de cohérence** : pas de clause comme `annee < 200 and annee > 2001`

Si OK, traduction en une expression algébrique, plus "opérationnelle" que SQL.

Retenir

Toute requête SQL se réécrit en une expression de l'algèbre.

(Pas tout à fait vrai : `group by`, `having`, `order by`. Oublions.)

# Exemple de réécriture

On prend comme exemple : le titre du film paru en 1958, où l'un des acteurs joue le rôle de John Ferguson.

En SQL :

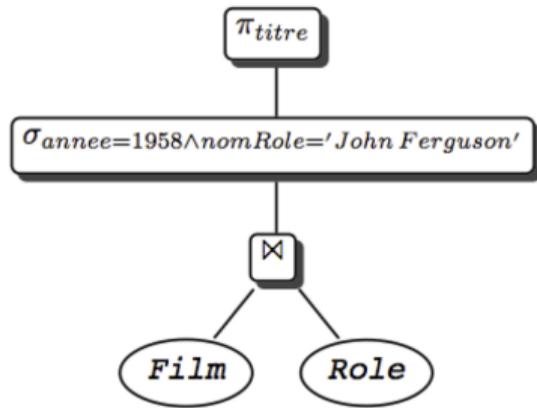
```
select titre
from   Film f, Role r
where  nom_role = 'John Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

En algèbre :

$$\pi_{titre}(\sigma_{annee=1958 \wedge nom\_role='John\ Ferguson'}(Film \bowtie_{id=id\_film} Role))$$

# Le Plan d'Exécution Logique (PEL)

L'expression algébrique 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.