SGBD-R : traitement de requêtes

Le modèle d'exécution

Guillaume Raschia — Nantes Université

originaux de Philippe Rigaux, CNAM

Dernière mise-à-jour : 5 septembre 2022

Plan de la session

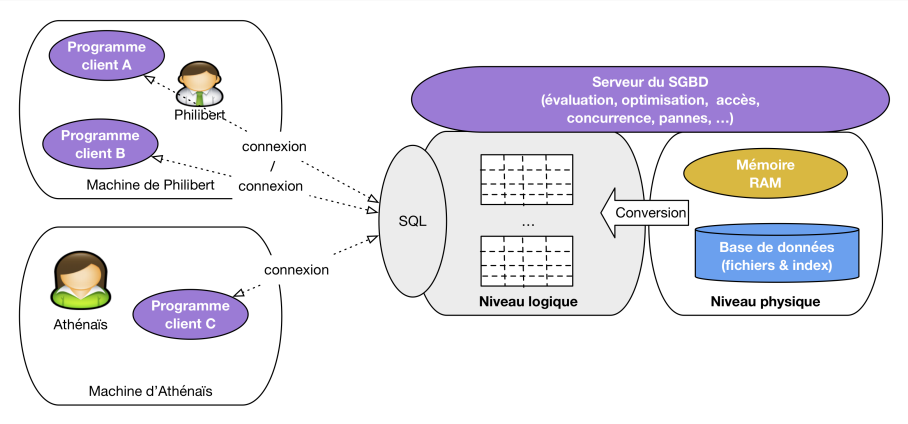
Rappels

Les itérateurs

Les opérateurs

Rappels

Vue d'ensemble d'un système relationnel



Retenir : indépendance entre le niveau logique et le niveau physique. Condition nécessaire de l'administration des SGBD.

Fonctionnalités d'un SGBD

Stockage : les données sont stockées de manière ordonnée sur des supports persistants

Indexation : des structures de données permettent des accès très rapides à la base

Interrogation et mise à jour : **les requêtes de haut niveau d'abstraction (SQL) sont transcrites en algorithmes efficaces**

Partage / Concurrence d'accès : de très nombreuses applications peuvent accéder simultanément à une même base

Sécurité : contrôle des accès et gestion des pannes logicielles ou matérielles

Rappels sur le modèle relationnel

Les données sont structurées en tables (relations) et lignes (nuplets). La ligne est l'unité d'information.

idArtiste	nom	prénom	annéeNaissance
100	Eastwood	Clint	1930
101	Hackman	Gene	1930
102	Scott	Tony	1930
103	Smith	Will	1968

Les données sont structurées et contraintes par un schéma.

```
create table Artiste ( idArtiste      integer not null,
                       nom            varchar (30) not null,
                       prénom         varchar (30) not null,
                       annéeNaissance integer,
                       primary key (idArtiste) );
```

Clé primaire, clé étrangère

Les tables sont liées par des valeurs partagées entre clé primaire et clé étrangère.

idFilm	titre	année	genre	*idRéalisateur*	*codePays*
20	Impitoyable	1992	Western	100	USA
21	Ennemi d'état	1998	Action	102	USA

```
create table Film ( idFilm         integer not null,
                    titre          varchar (80) not null,
                    année          integer not null,
                    genre          varchar (20) not null,
                    idRéalisateur  integer,
                    codePays       varchar (4),
                    primary key (idFilm),
                    foreign key (idRéalisateur) references Artiste(idArtiste),
                    foreign key (codePays) references Pays(code) );
```

Les langages

Le modèle relationnel s'appuie sur deux langages très différents mais équivalents en pouvoir expressif.

- SQL a pour fondement la logique des prédicats.
  - C'est un langage déclaratif qui indique ce que l'on veut obtenir sans spécifier comment on calcule le résultat.
- L'algèbre relationnelle est un langage fonctionnel composé d'opérateurs.
  - L'algèbre permet de construire des plans d'exécutions indiquant comment on évalue une requête.

Évaluation et optimisation

Le SGBD transforme une requête du niveau logique (SQL) en un programme d'évaluation algébrique au niveau physique.

Le langage SQL : supposé connu !

Une sélection

```
select titre
from Film
where année = 2016
```

Une jointure

```
select f.titre, a.prénom, a.nom
from Film as f, Artiste as a
where f.idRéalisateur = a.idArtiste
and   année = 2016
```

La méthode de calcul n'est jamais spécifiée.

Algèbre relationnelle : supposée connue également

Opérateurs qui prennent une ou deux tables en entrée et produisent une table en sortie.

- Sélection (σ) : restriction sur les valeurs des nuplets
- Projection (π) : suppression de colonnes (allège le résultat)
- Jointure ($\bowtie$) : fusionne deux ensembles sur condition
- Différence ($-$) : sous-ensemble non présent dans le second
- Union ($\cup$) : union des deux ensembles

Les opérateurs sont composables pour construire des expressions

$$\pi_{\text{titre}}(\sigma_{\text{année}=2016}(\text{Film}))$$

$$\pi_{\text{titre,prénom,nom}}(\sigma_{\text{année}=2016}(\text{Film}) \underset{\text{idRéalisateur}=\text{idArtiste}}{\bowtie} \text{Artiste})$$

À retenir

Suivre et comprendre un cours sur les aspects systèmes des SGBDs relationnels suppose des connaissances préalables

- Sur l'architecture des systèmes relationnels, et notamment l'indépendance entre niveau physique et niveau logique
- Sur les principes de conception des schémas relationnels et leurs contraintes
- Sur le langage SQL
- Et sur l'algèbre relationnelle

À réviser d'urgence si vous avez des doutes ou des lacunes.

Les itérateurs

Plan d’exécution et opérateurs

Définition (Plan d’exécution de requête)

un plan d’exécution est un arbre constitué d’opérateurs échangeant des flux de données.

Dans cette section

On va apprendre à implanter un moteur d’exécution pour (presque toutes) les requêtes SQL

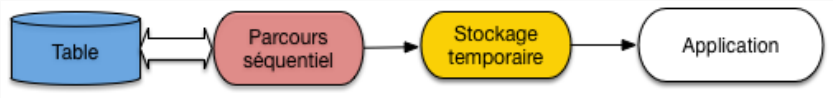
Caractéristiques de opérateurs

- Ils ont une forme générique : itérateurs
- Ils fournissent une tâche spécialisée (cf. l'algèbre relationnelle)
- Ils sont composables
- Ils peuvent être ou non bloquants

Un petit nombre suffit pour couvrir SQL !

Mode naïf : matérialisation

Dans ce mode, un opérateur calcule son résultat, puis le transmet.

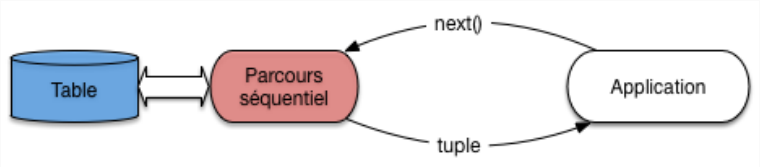


Deux inconvénients :

- Consomme de la mémoire.
- Introduit un temps de latence.

La bonne solution : pipeline

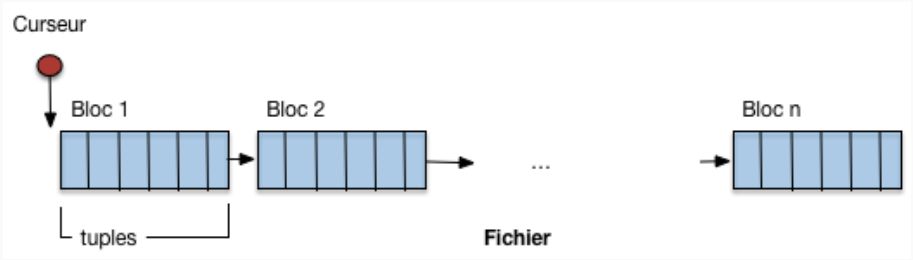
Le résultat est produit à la demande.



- Pas de stockage intermédiaire.
- Latence minimale.

Illustration : l'opérateur `FullScan`

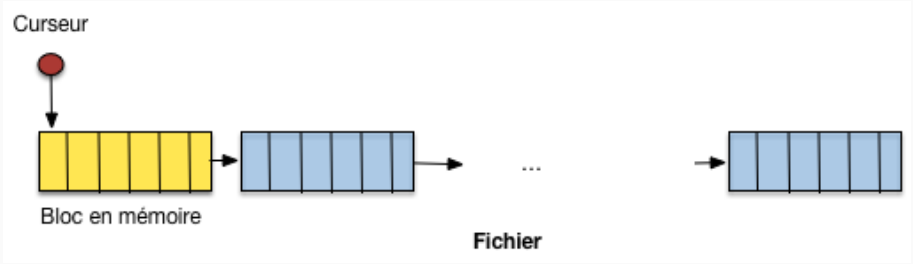
Au moment du `open()`, le curseur est positionné avant le premier nuplet.



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

Illustration : l'opérateur `FullScan`

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



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

Illustration : l'opérateur `FullScan`

Le deuxième `next()` avance d'un cran dans le parcours du bloc.

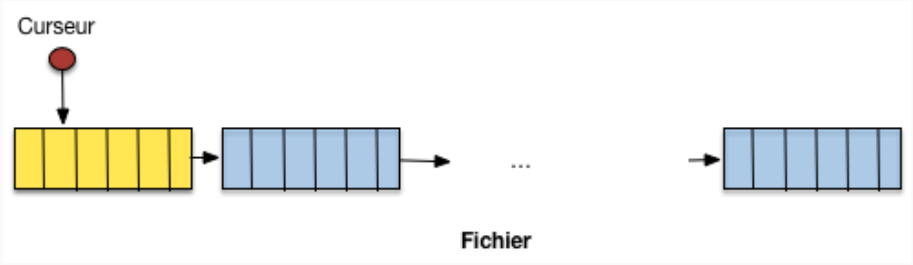


Illustration : l'opérateur `FullScan`

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

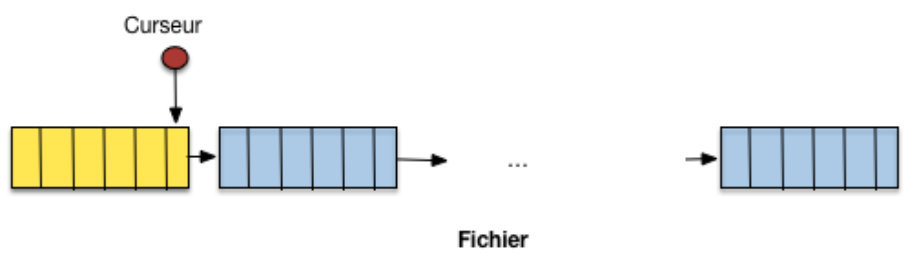


Illustration : l'opérateur `FullScan`

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

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

Exécuter une requête = avancer un curseur

```
select * from T
```

est implantée par :

```
# Parcours  de la table T
curseur = FullScan(T)   # création et initialisation
curseur.open()          # préparation des (res-)sources
while (nuplet := curseur.next()) is not None:
    # Traitement du nuplet
    continue
```

Ce mécanisme d'itération est général pour l'exécution de requêtes.

Chaînage des itérateurs

Un arbre d'exécution de requête combine à la volée les itérateurs de différents opérateurs.

- L'application « branche un aspirateur » sur le dernier opérateur de la requête et tire à la demande les nuplets.
- À son tour, l'opérateur tire les nuplets de chacune de ses sources pour répondre à la demande, et ainsi de suite jusqu'aux données.

Opérateur bloquant

Tous les opérateurs peuvent-ils fonctionner en pipeline ?

```
select min(date) from T
```

On ne peut pas produire un nuplet avant d'avoir examiné toute la table.

Il faut alors introduire un opérateur bloquant, avec une latence forte.

Avec un opérateur bloquant, on additionne le temps d'exécution de la requête et le temps de traitement de l'application.

Résumé : opérateurs d'exécution

Principes essentiels :

- Itération : dans tous les cas, un opérateur produit les nuplets à la demande.
- Pipeline : si possible, le résultat est calculé au fur et à mesure.
- Matérialisation : parfois le résultat intermédiaire doit être calculé et stocké.

Bien distinguer

- Temps de réponse : temps pour obtenir le premier nuplet.
- Temps d'exécution : temps pour obtenir tous les nuplets.

Les opérateurs

Opérateur = itérateur

Tout opérateur est implanté sous forme d'un itérateur. Trois fonctions :

- `open` : initialise les ressources et positionne le curseur ;
- `next` : fournit l'enregistrement courant et se place sur le suivant ;
- `close` : libère les ressources.

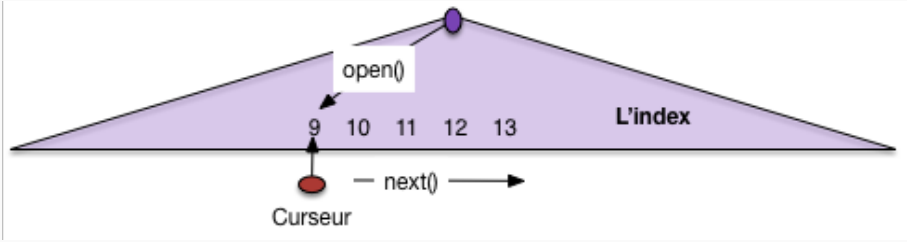
Échanges :

- Un itérateur consomme des nuplets d'autres itérateurs source.
- Un itérateur produit des nuplets pour un autre itérateur (ou pour l'application).

Exemple : parcours d'index (`IndexScan`)

Rappel : index = arbre B+.

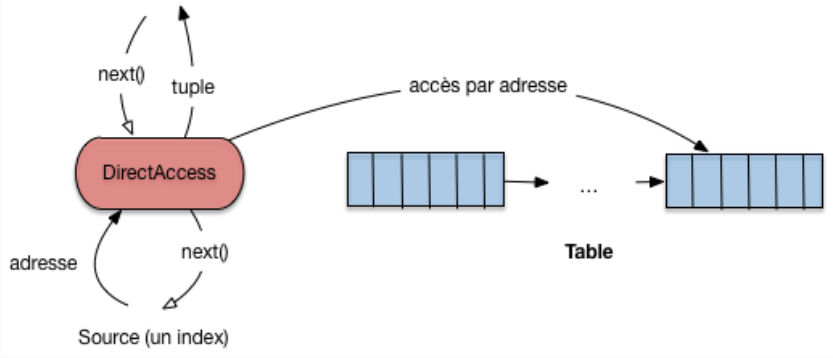
- Pendant le `open()` : parcours de la racine vers la feuille.
- À chaque appel à `next()` : parcours en séquence des feuilles.



Efficacité : très efficace, quelques lectures logiques (index en mémoire)

Accès par adresse : `DirectAccess`

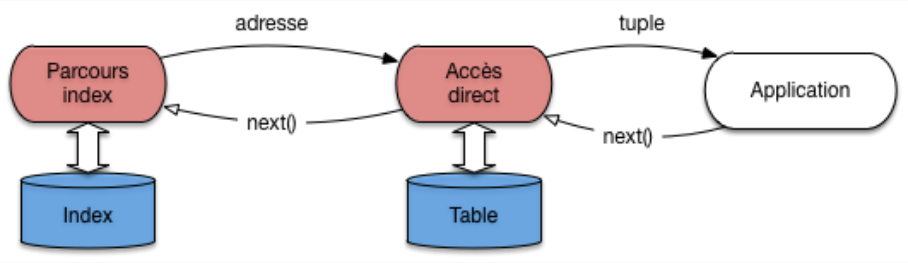
- Pendant le `open()` : rien à faire.
- À chaque appel à `next()` : on reçoit une adresse, on produit un nuplet.



Très efficace : un accès au bloc, souvent en mémoire.

Plan d'exécution

Un plan d'exécution connecte les opérateurs. Ici, recherche avec index.



Le pipeline est effectif.

Un exemple de base

Nous allons étudier les plans permettant d'exécuter les requêtes mono-table.

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

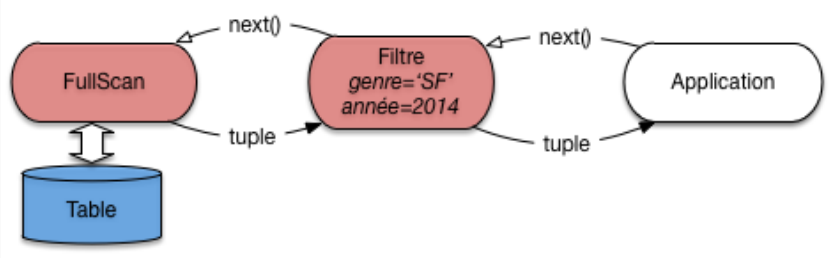
De quels opérateurs a-t-on besoin ?

- [`FullScan`] : parcours séquentiel de la table (déjà vu).
- [`IndexScan`] : parcours d'un index (si disponible).
- [`DirectAccess`] : accès par adresse à un nuplet.
- [`Filter`] : test de la condition.

Nous obtenons deux plans d'exécution possibles.

Premier plan d'exécution : sans index

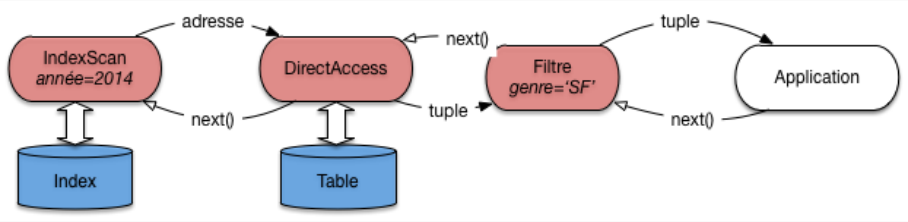
```
select titre from Film where genre='SF' and annee = 2014
```



N'utilise pas d'index

Second plan d'exécution : avec index

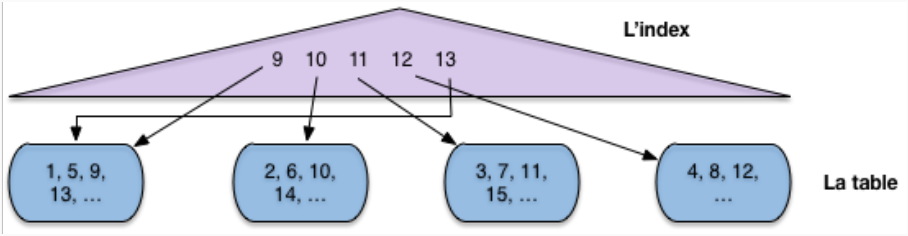
```
select titre from Film where genre='SF' and annee = 2014
```



Utilise un index sur l'année.

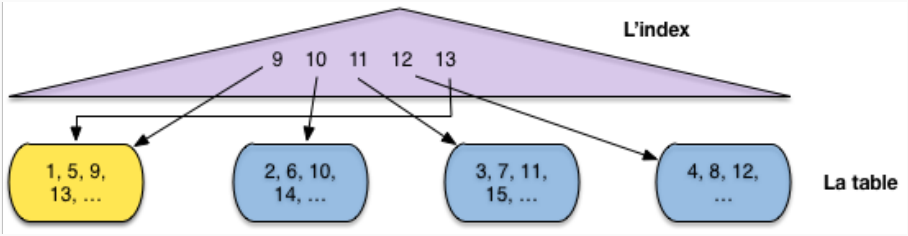
Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



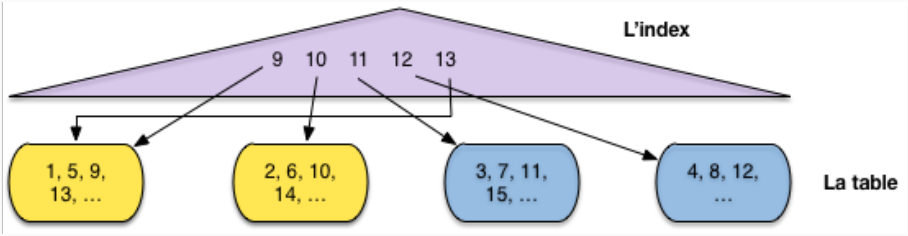
Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



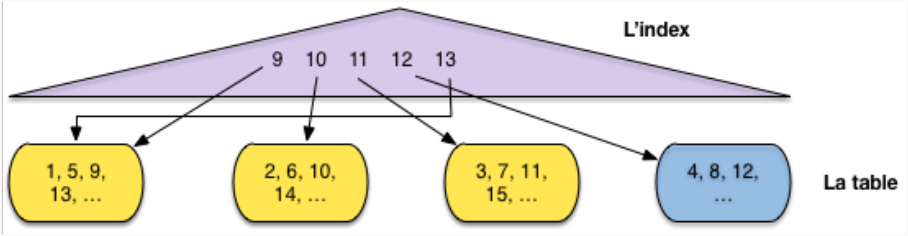
Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



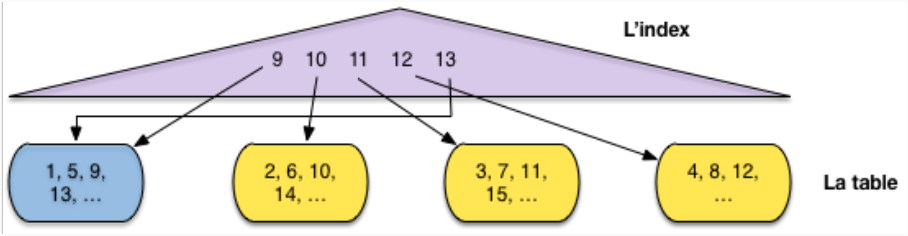
Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



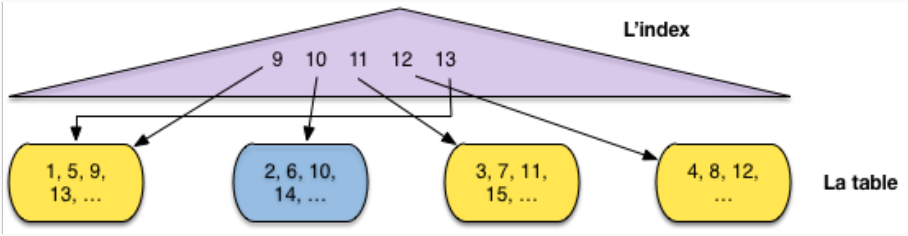
Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.



Index ou pas index ?

Recherche des nuplets entre 9 et 13, avec 3 blocs en mémoire.

Avec l'index, il faut lire 5 blocs ! Parcours séquentiel bien préférable.

Un cas extrême !

En pratique : le SGBD décide en fonction des statistiques et des ressources disponibles.

Opérateur de sélection

Très simple : filtre les nuplets fournis par la source.

```
# Fonction next de l'opérateur Filter
def next(cond):
    # On consomme un nuplet de la source (source.next())
    # tant que la condition n'est pas satisfaite
    # et la source intégralement parcourue
    while (nuplet := source.next()) is not None
        and test(nuplet, cond) is False:
        continue
    return nuplet
```

Pour compléter

Un plan à exécuter avec index.

```
select * from Film
where  idFilm = 20
and    titre = 'Vertigo'
```

Un plan à exécuter sans index.

```
select * from Film
where  idFilm = 20
or     titre = 'Vertigo'
```

Résumé : plans pour requêtes mono-tables

Permier aperçu de l'optimisation

- Le système a le choix entre plusieurs plans possibles.
- Distinguer le plus efficace n'est pas toujours trivial.
- Le choix peut changer selon le contexte.

Les méthodes d'accès **FullScan** et **IndexScan+DirectAccess**, ainsi que l'opérateur **Filter** sont connus.

L'opérateur de **projection** est trivial, bien souvent intégré aux autres dans les plans d'exécution de requêtes.

Il ne manque que le **tri** et la **jointure** pour un moteur d'exécution de requêtes conjonctives. Pas si compliqué !