SGBD – Optimisation de requête

**Intro :**

-Attributs -> champs(fields) de longueurs fixes ou variables

-Tupes -> enregistrements (record) de longueurs fixes ou variables

-Blocs physiques

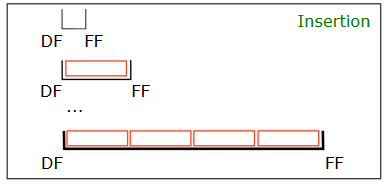
Dictionnaire

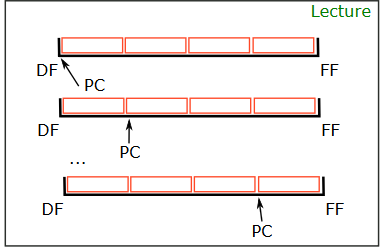
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | T1 | T2 | T3 |  | Tuple t3 | Tuple t2 |
|  | T1 | T2 | T3 |  | Tuple  t3 | Tuple  t2 |
|  | T1 | T2 | T3 |  | Tuple t3 | Tuple t2 |

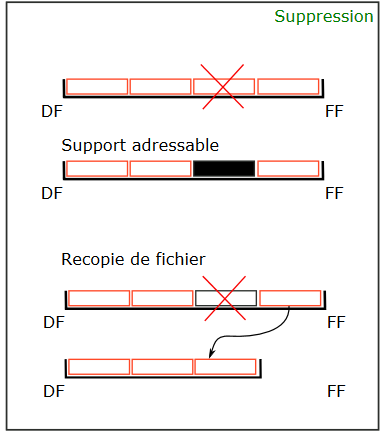
-Fichiers physiques

**Index primaires :**

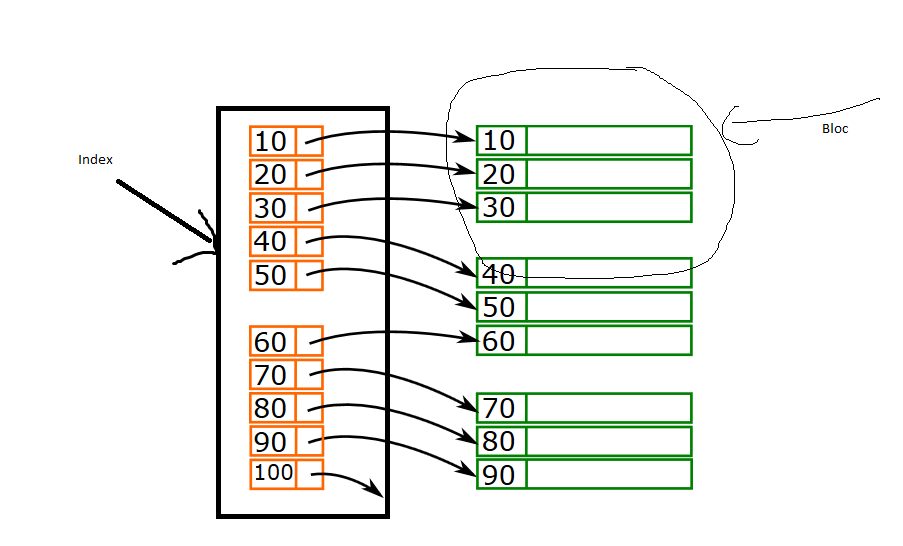
-Organisation séquentielle en Mémoire secondaire





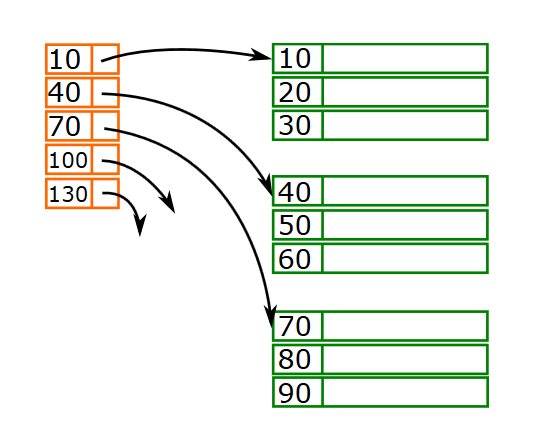


Dense index :



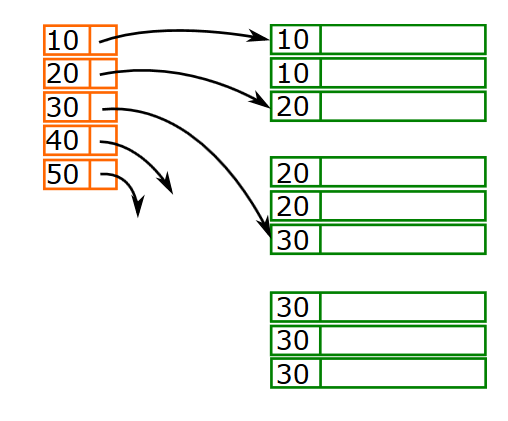
Sparse index :

On ne garde que les valeurs de début de bloc en index

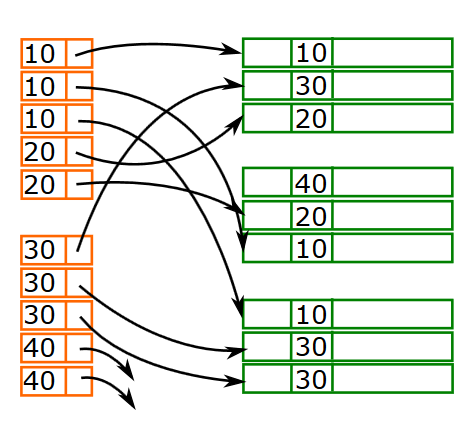


Index sur clés à valeurs non uniques (ex : n°commande dans LIGNE\_COMMANDE) :

L’index pointe sur la première occurrence



**Index secondaires :**

L’ordre d’insertion des données ne se fait pas dans un ordre précis : insertion d’étudiants dans une table, l’index porte sur le nom, il est donc désordonné****

**B-arbres :**

Maintien automatique du nombre de niveaux de l’arbre

Pas de dépassement de la taille des blocs

Arbre parfaitement équilibré :

* Toutes les feuilles sont au même niveau
* Tous les chemins depuis la racine jusqu’à une fueille sont de même longueur
* Temps d’accès pratiquement constant

Il existe plusieurs versions

Les clés sont des copies des clés stockées dans le fichier de la table

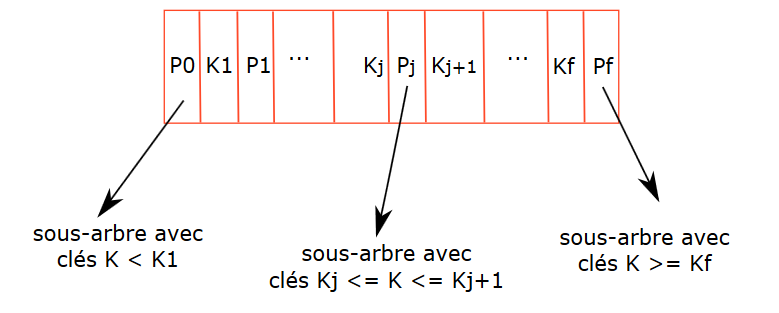
Les clés sont réparties dans les feuilles et ordonnées de gauche à droite

Dans le nœud racine il y a au moins 2 pointeurs vers les niveaux inférieurs

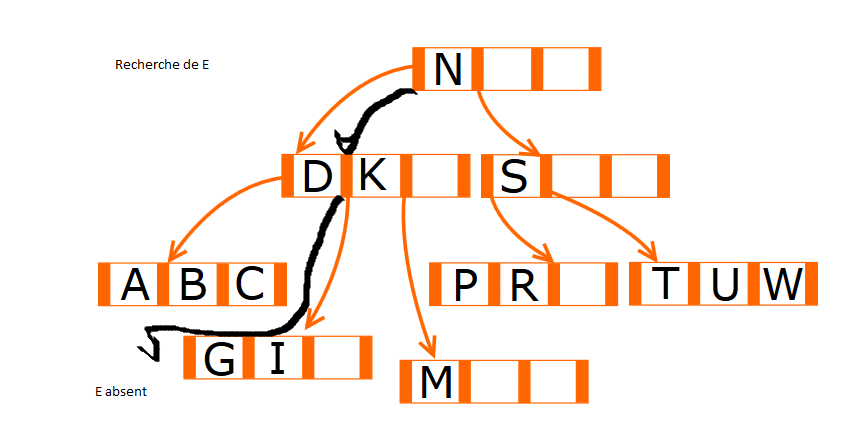
Chaque nœud sauf la racine contient au plus n clés (et au moins n/2)

Chaque nœud qui a x clés possède x+1 pointeurs vers le niveau inférieur (sauf les feuilles)

Structure d’un nœud :



Exemple :



Organisation des B-arbres :

Un nœud de l’arbre peut contenir :

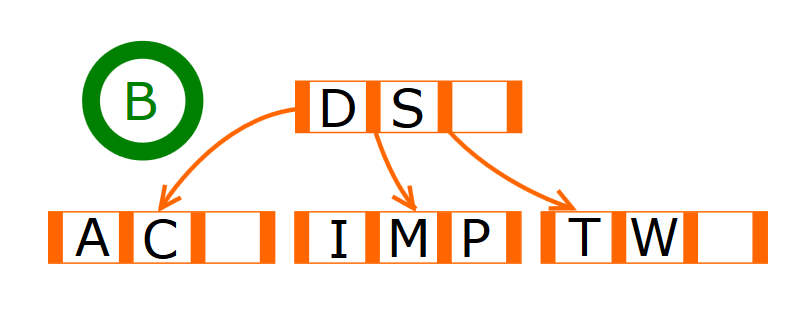
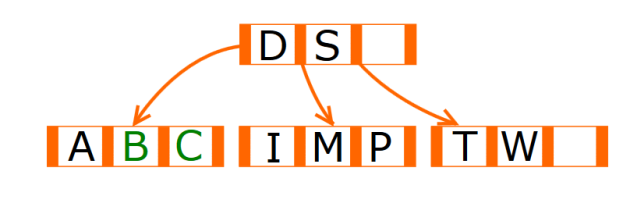
* Le tuple en entier
* Un pointeur sur le bloc qui contient le tuple

L’index peut être dense ou sparse (voir au dessus)

**Insertion :**

Parcours descendant pour trouver où l’insertion doit avoir lieu (pour garder l’équilibre de l’arbre

Ajout de la clé dans la feuille (peut nécessiter un décalage vers la droite

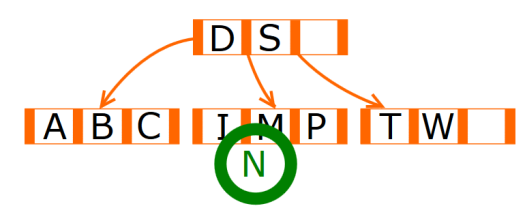
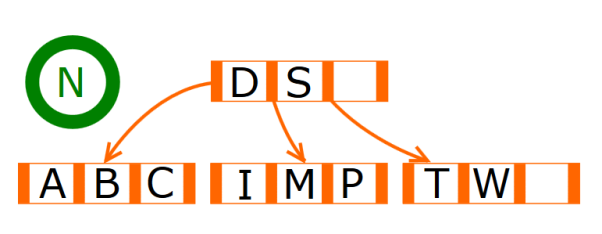
 

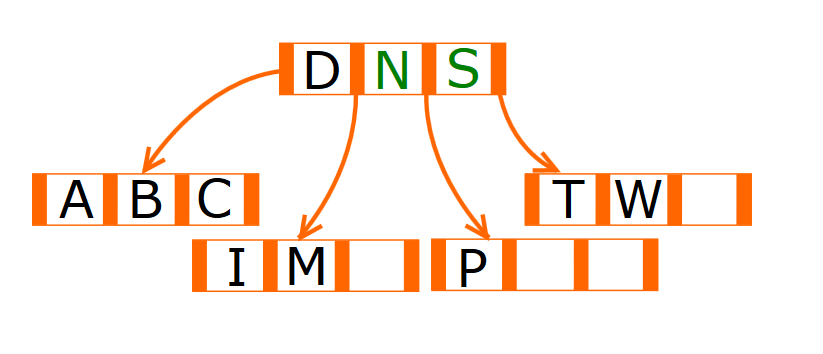
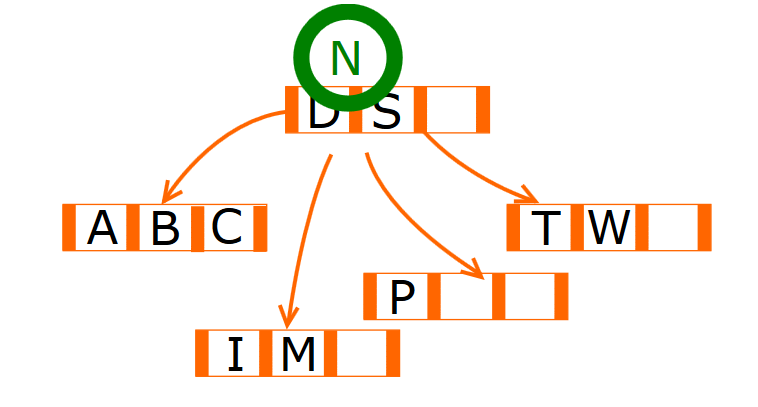
Insertion dans une feuille pleine :

L’éclater en deux en ajoutant une feuille à droite

Répartir les clés sur les deux feuilles

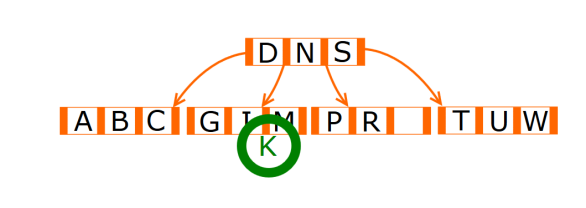
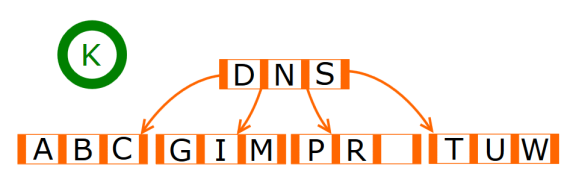
Remonter le pivot (valeur médiane) dans le nœud père (processus récursif)

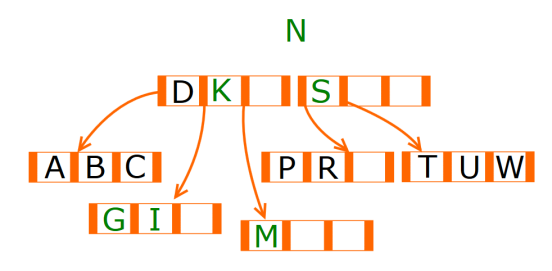
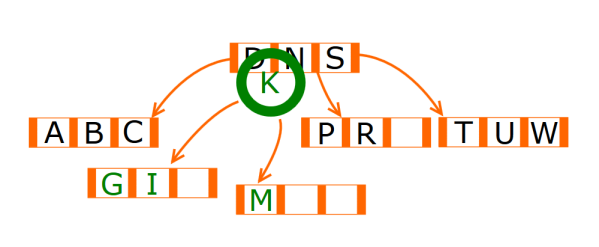


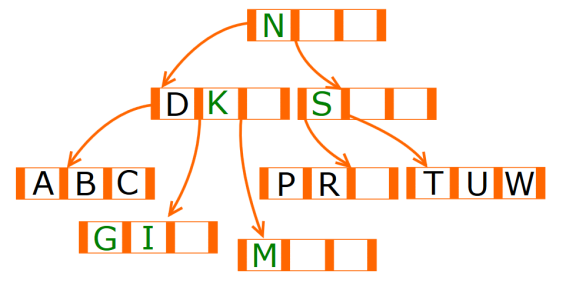


Feuilles pleines :

Remontée d’un pivot dans le nœud père existant(racine éclatée ) : Création d’une nouvelle racine



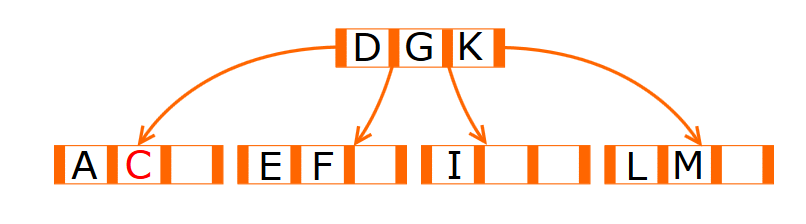
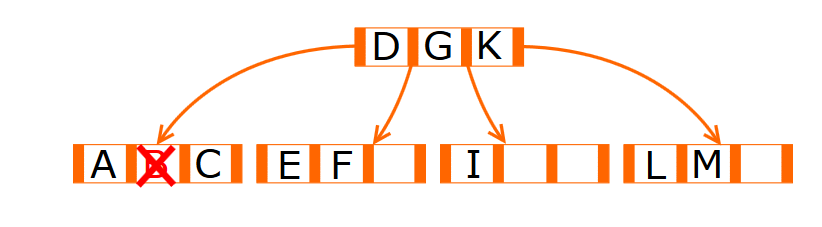




**Suppression :**

Parcours descendant pour trouver dans quelle feuille la suppression doit avoir lieu

Suppression d’une clé dans une feuille (peut nécessiter un décalage vers la gauche)



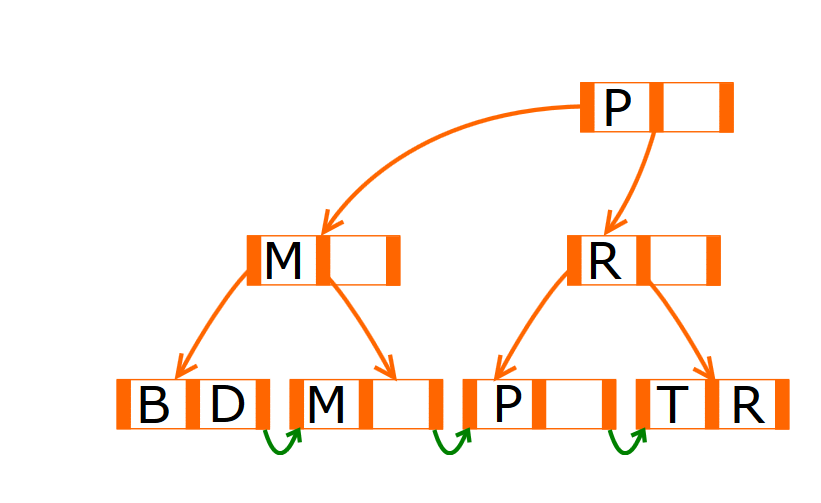
Suppression d’une clé qui n’est pas dans une feuille

Echange de la clé avec son successeur immédiat (clé qui se trouve dans une feuille)

Suppression de la clé (ramenée dans une feuille)

//TODO INSERT DATA

**Arbre B+ :**

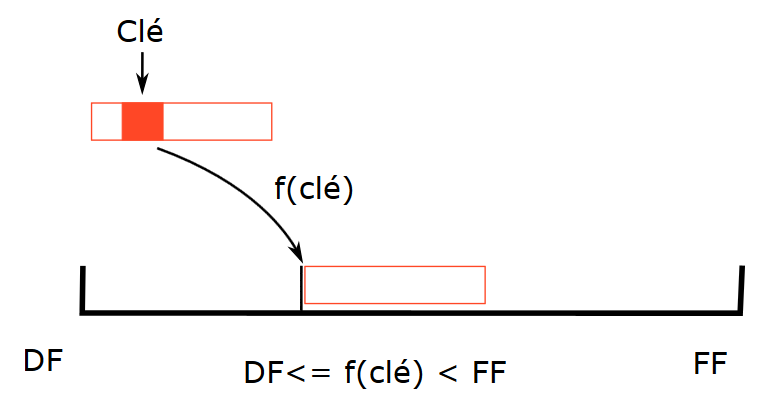


Feuilles chaînées (travail sur des intervalles par exemple)

**H-Tables :**

Notion de clé de hashage

!= Clé relationnelle



Calcul de l’adresse mémoire en fonction du hash de la clé

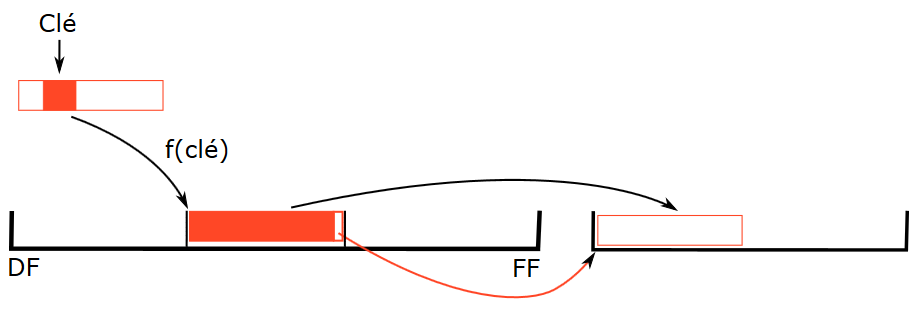
Réservation de **B** blocs en mémoire

Fonction de hashage qui renvoie une valeur entre 0 et B-1

* Opération de folding : transformation de la clé en une valeur entière positive distribuée de façon uniforme sur un large intervalle
* Opération de hashage : transformation de la valeur trouvée en une valeur entière comprise entre 0 et b-1

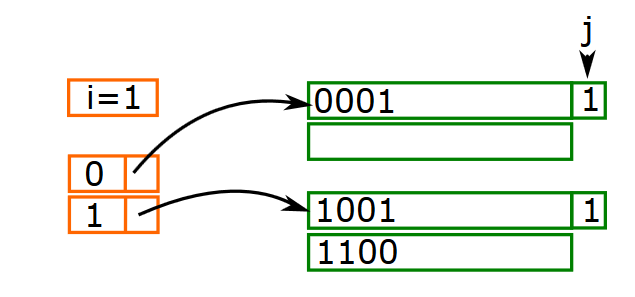
Avantage : recherche de la clé directement à partir de sa valeur

Inconvénients : collisions et débordements



FF : pointeur sur le bloc suivant, si le tuple est plus grand que la valeur allouée par défaut

Tables de hashage extensibles (index) :

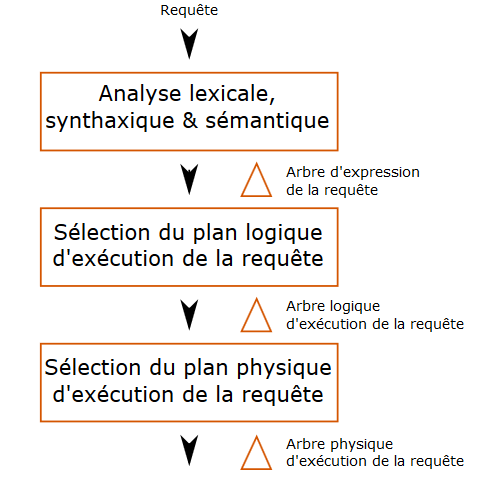


Insertion :

* Si **j < i**, dédoublement du bloc
* Sinon incrémenter la valeur de **i** et augmenter la taille de l’index

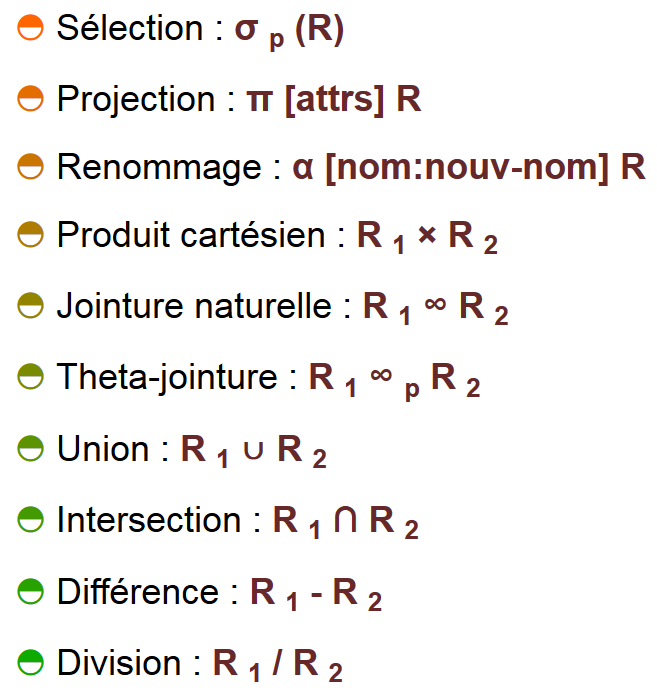
Optimisation de requêtes

Introduction :

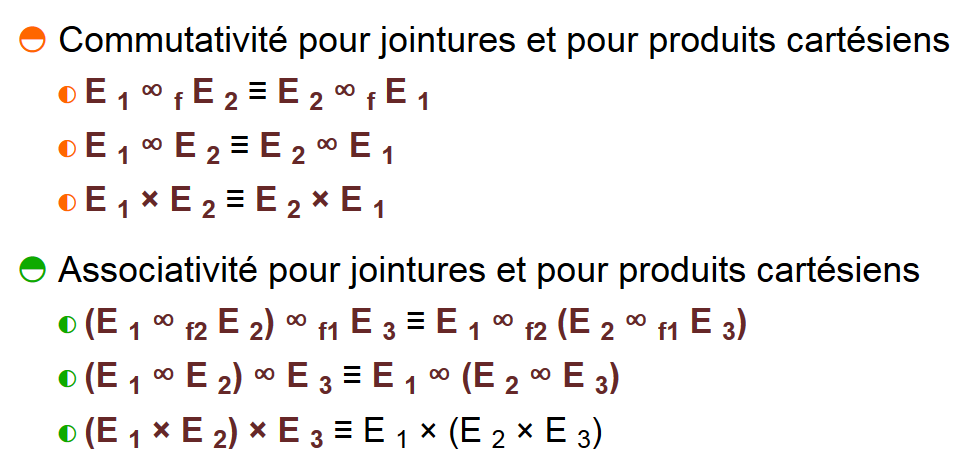


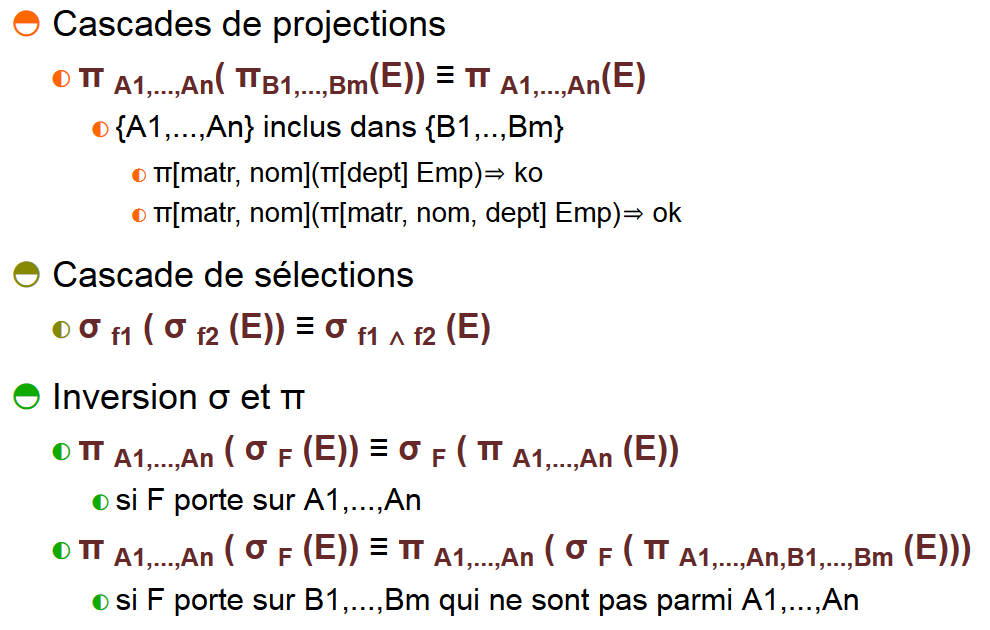
Recherche des plans logiques d’exécutions possibles

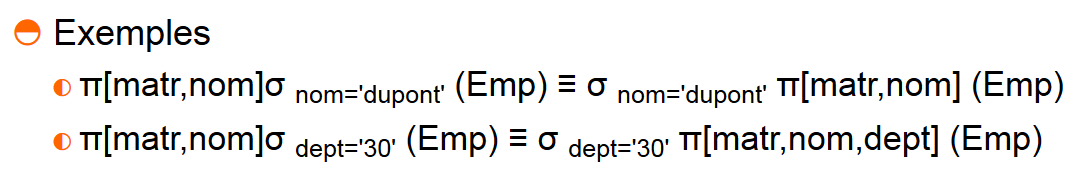
Rappel notation algébrique

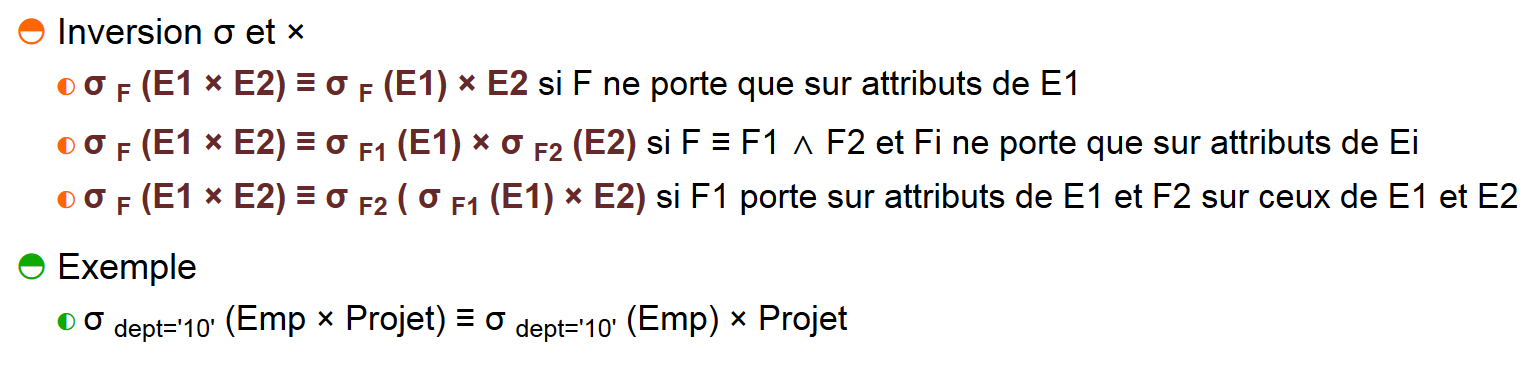


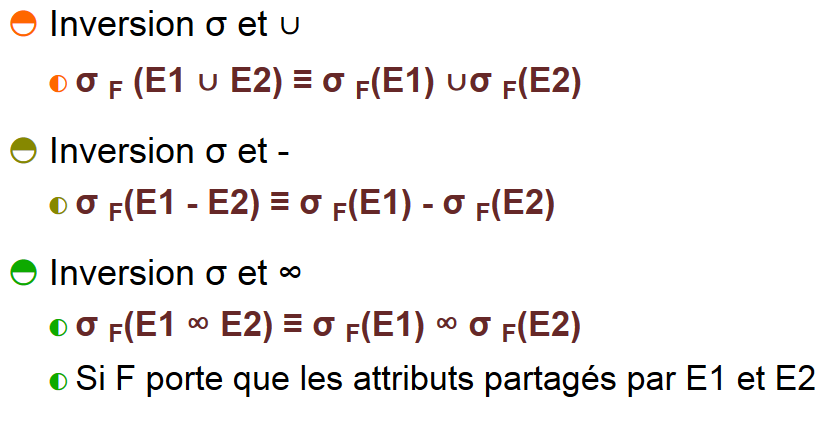
Propriétés des opérateurs :

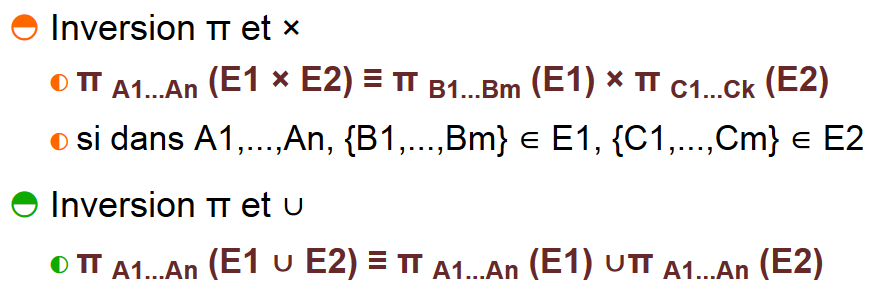




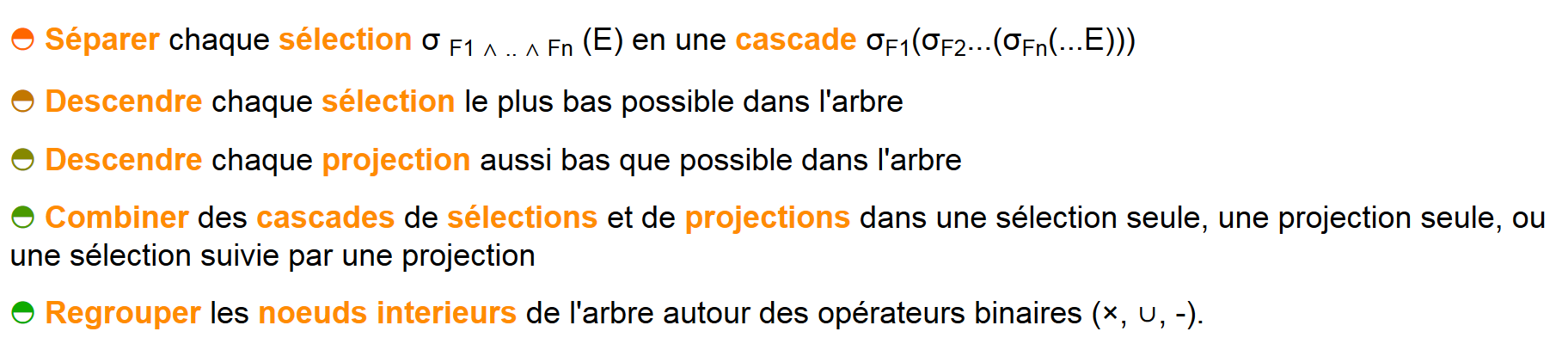




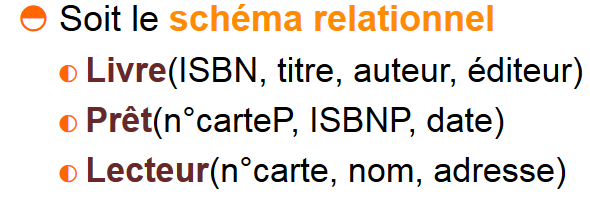


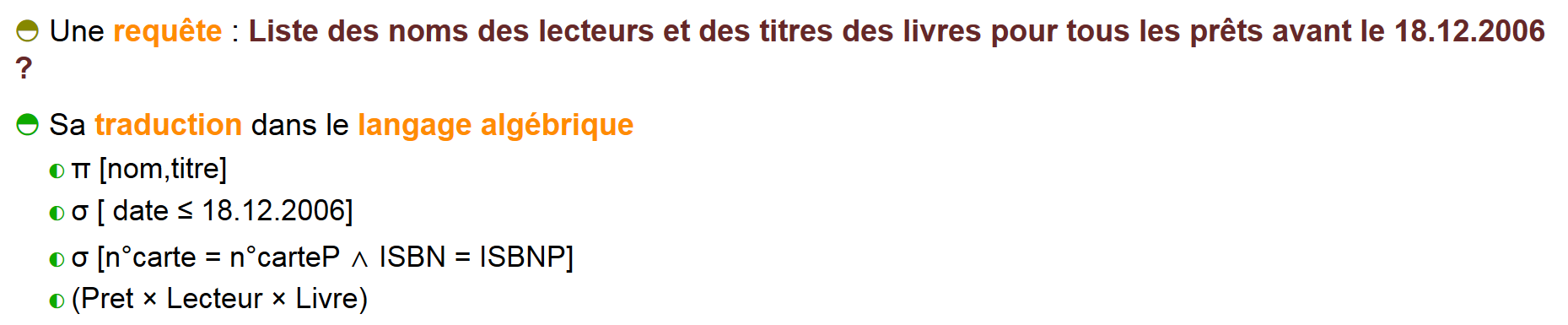


Construction d’un plan logique d’exécution d’une requête

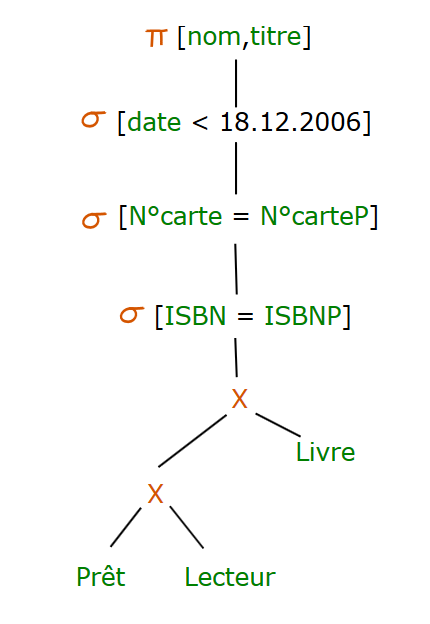


Exemple :





Après décomposition des sélections



Après descente des sélections

