



# **CHAPITRE 3**

## **Mémoire Relationnelle**

# CHAPITRE 3: Mémoire Relationnelle

## Plan

### 1 INTRODUCTION

### 2 RAPPELS SUR LES METHODES CLASSIQUES DE STOCKAGE

2.1 Le Fichier comme un « tas de données »

2.2 Le hachage ou adressage associatif

2.3 Fichiers indexés



### 3 INDEX EN B-ARBRE

3.1 Définitions

3.2 Algorithme d'insertion

3.3 Algorithme de suppression

### 4 HACHAGE VIRTUEL

4.1 Introduction

4.2 hachage virtuel HV1

# CHAPITRE 3: Mémoire Relationnelle

## Plan

### 1 INTRODUCTION

### 2 RAPPELS SUR LES METHODES CLASSIQUES DE STOCKAGE

2.1 Le Fichier comme un « tas de données »

2.2 Le hachage ou adressage associatif

2.3 Fichiers indexés



### 3 INDEX EN B-ARBRE

3.1 Définitions

3.2 Algorithme d'insertion

3.3 Algorithme de suppression

### 4 HACHAGE VIRTUEL

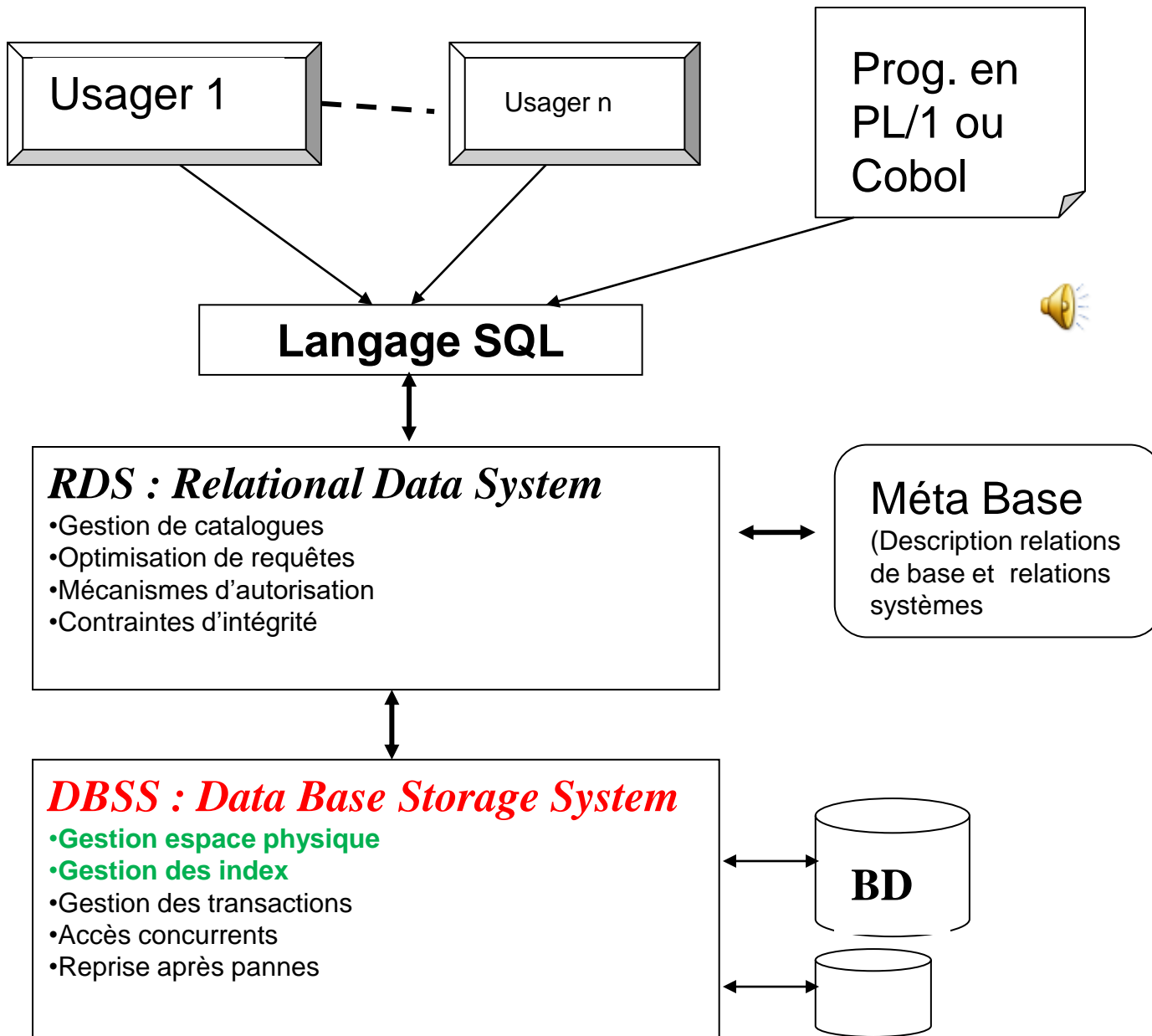
4.1 Introduction

4.2 hachage virtuel HV1

## Introduction



Nous nous intéressons dans ce chapitre à la couche du SGBD qui accède à la BD, c'est-à-dire au système de stockage appelé **la mémoire relationnelle**.



# Introduction

Nous nous intéressons dans ce chapitre à la couche du SGBD qui accède à la BD, c'est-à-dire au système de stockage appelé **la mémoire relationnelle**.

Cette couche présente quatre **fonctions principales** :



**1. Présenter une vision relationnelle des fichiers** qui constituent la BD. Nous rappelons qu'un fichier est un ensemble d'enregistrements composés eux-mêmes de champs. En première approximation, une relation peut être vue comme un fichier, et chaque tuple correspond à un enregistrement.

# Introduction

Nous nous intéressons dans ce chapitre à la couche du SGBD qui accède à la BD, c'est-à-dire au système de stockage appelé **la mémoire relationnelle**.

Cette couche présente quatre **fonctions principales** :

**1. Présenter une vision relationnelle des fichiers** qui constituent la BD. Nous rappelons qu'un fichier est un ensemble d'enregistrements composés eux-mêmes de champs. En première approximation, une relation peut être vue comme un fichier, et chaque tuple correspond à un enregistrement.

**2. Gérer la mémoire centrale (MC) :** les *relations* sont **découpées** et stockées par **page de taille fixe en mémoire secondaire (MS)**, lesquelles sont amenées en MC à la demande, grâce à des mécanismes de mémoire virtuelle. L'espace de stockage en **MS** est logiquement divisé en **segments**. Un **segment** est un **espace virtuel** de stockage qui **se compose physiquement de plusieurs pages**. Nous rappelons que la page est l'unité de transfert entre MC et MS. Un fichier est stocké dans un seul segment et peut s'étendre donc, sur plusieurs pages.



# Introduction

Nous nous intéressons dans ce chapitre à la couche du SGBD qui accède à la BD, c'est-à-dire au système de stockage appelé **la mémoire relationnelle**.

Cette couche présente quatre **fonctions principales** :

**1. Présenter une vision relationnelle des fichiers** qui constituent la BD. Nous rappelons qu'un fichier est un ensemble d'enregistrements composés eux-mêmes de champs. En première approximation, une relation peut être vue comme un fichier, et chaque tuple correspond à un enregistrement.

**2. Gérer la mémoire centrale (MC)** : les **relations** sont **découpées** et stockées par **page de taille fixe en mémoire secondaire (MS)**, lesquelles sont amenées en MC à la demande, grâce à des mécanismes de mémoire virtuelle. L'espace de stockage en **MS** est logiquement divisé en **segments**. Un **segment** est un **espace virtuel** de stockage qui **se compose physiquement de plusieurs pages**. Nous rappelons que la page est l'unité de transfert entre MC et MS. Un fichier est stocké dans un seul segment et peut s'étendre donc, sur plusieurs pages.

**3. Gérer les différentes relations de la BD**, c'est-à-dire les relations de **base**, les **index** appelés encore **relations inverses**, les **vues**, les différentes relations **catalogues** permettant de les décrire.





**4. Etablir des méthodes d'accès :** nous distinguons les méthodes classiques (les méthodes indexées, le hachage etc.) et des méthodes plus particulières à savoir le hachage virtuel et les arbres balancés B-arbres.

Les opérations que nous pouvons effectuer sur le fichier sont :



- Insérer un enregistrement
- Supprimer un enregistrement
- Modifier un enregistrement.

# Rappels sur les méthodes classiques de stockage

## 1. Le fichier comme un « tas de données »

Dans cette technique, les **enregistrements** sont placés à **la queue leu leu** dans des pages successives. Un enregistrement **n'est pas à cheval sur deux pages**. Le seul accès aux données dans le cas de la recherche d'un enregistrement est un **balayage séquentiel** des tuples. Cette opération est donc très coûteuse.



# Rappels sur les méthodes classiques de stockage

## 1. Le fichier comme un « tas de données »

Dans cette technique, les **enregistrements** sont placés à **la queue** **leu leu** dans des pages successives. Un enregistrement **n'est pas à cheval sur deux pages**. Le seul accès aux données dans le cas de la recherche d'un enregistrement est un **balayage séquentiel** des tuples. Cette opération est donc très coûteuse.

**L'insertion** d'un nouvel enregistrement s'effectue dans la **dernière page**. Si cette dernière est saturée, alors une nouvelle page est allouée et l'enregistrement est inséré.



# Rappels sur les méthodes classiques de stockage

## 1. Le fichier comme un « tas de données »

Dans cette technique, les **enregistrements** sont placés à la **queue** **leu leu** dans des pages successives. Un enregistrement **n'est pas à cheval sur deux pages**. Le seul accès aux données dans le cas de la recherche d'un enregistrement est un **balayage séquentiel** des tuples. Cette opération est donc très coûteuse.

**L'insertion** d'un nouvel enregistrement s'effectue dans la **dernière page**. Si cette dernière est saturée, alors une nouvelle page est allouée et l'enregistrement est inséré.

La **suppression** est *assurée logiquement* grâce à un *indicateur de suppression*.

Cette organisation est généralement *implémentée par défaut* dans les SGBD.



# Rappels sur les méthodes classiques de stockage

## 2. LE HACHAGE OU ADRESSAGE ASSOCIATIF

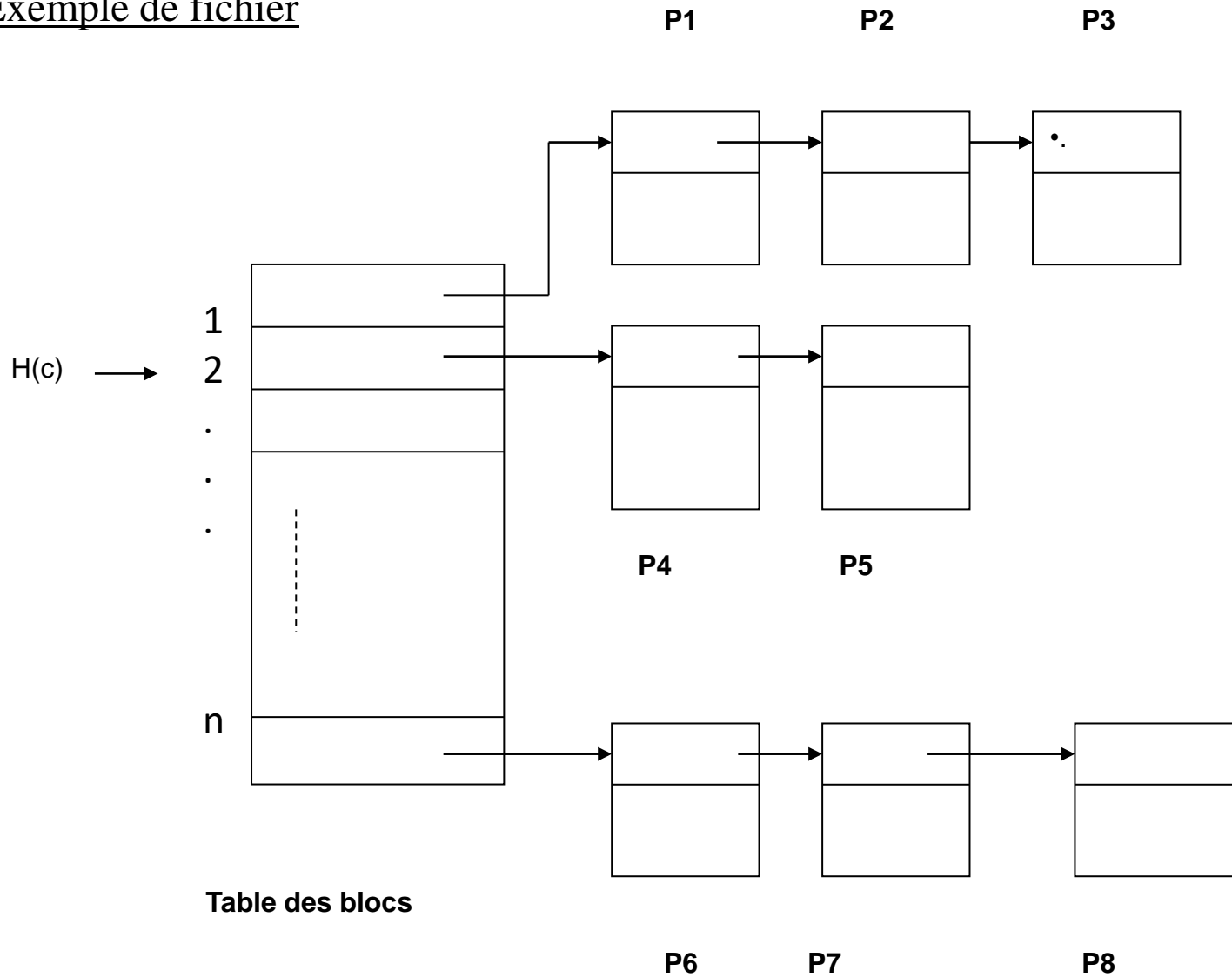
**Principe :**

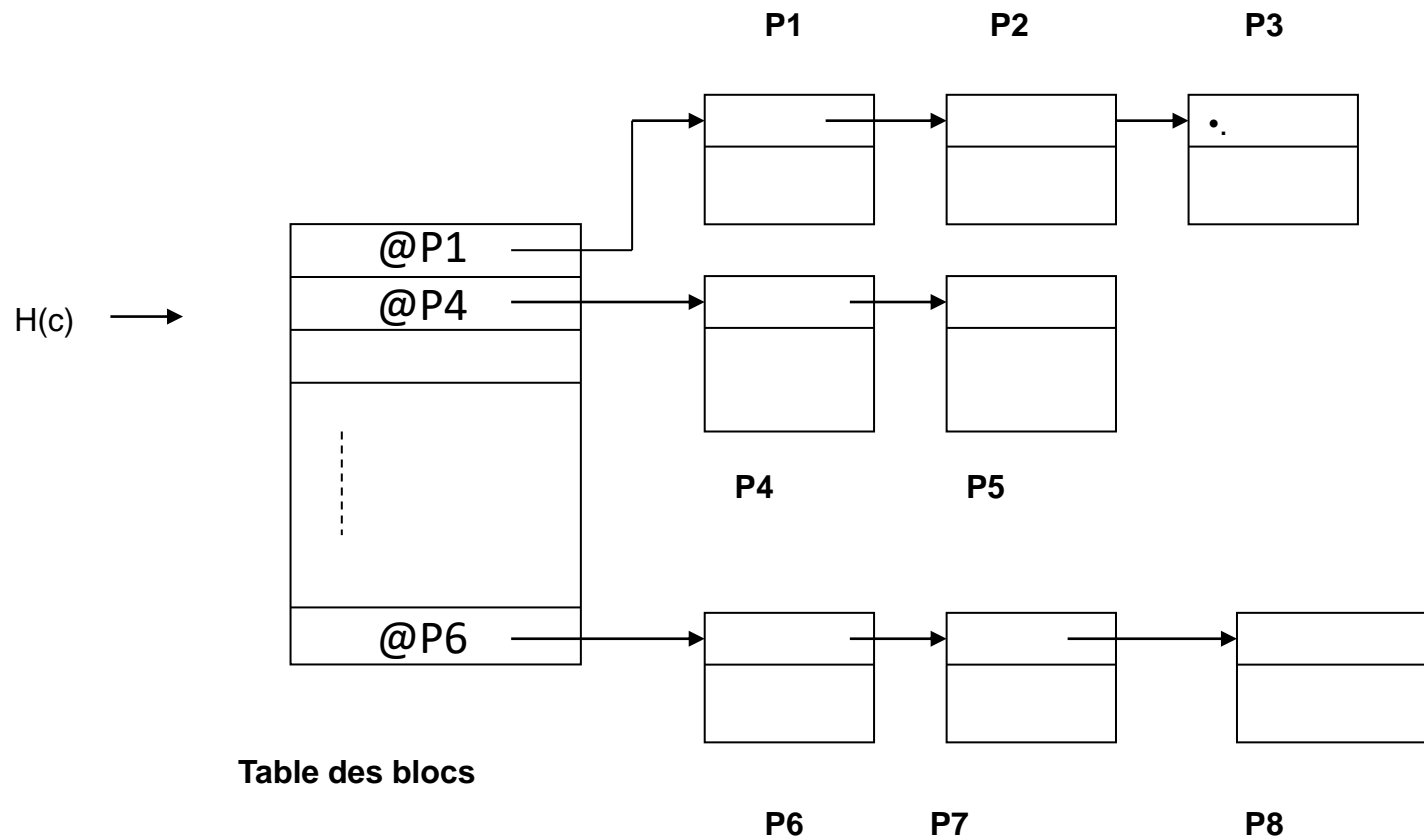


Disposer d'une fonction de hachage  $h(c)$  qui permet de calculer un **numéro de bloc ou paquet**, contenant un ensemble de pages, à partir d'une clé.

La recherche de l'enregistrement s'effectue ensuite séquentiellement dans le bloc.

## Exemple de fichier





**Explication** : La table des blocs va contenir  $n+1$  pointeurs vers  $n+1$  blocs. Chaque pointeur est l'adresse de la première page du bloc. Un bloc peut contenir une à plusieurs pages chaînées entre elles.

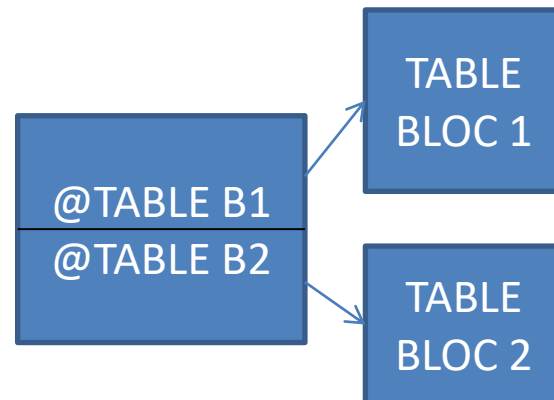
La fonction de hachage permet de retrouver à partir de la clé le numéro du bloc qui va contenir l'enregistrement.



### Exp:

$H(c)$  est une fonction de hachage  $\text{Mod}[100]$  Donc pour la clé 203. On prend l'entrée N° 3 de la table des blocs.

**Remarque :** la **table des blocs** peut être résidente en **MC** si elle est petite. Dans le cas contraire (définie sur plusieurs pages), il faudra la ramener de la **MS** par partie (par page) en appliquant la fonction de hachage.



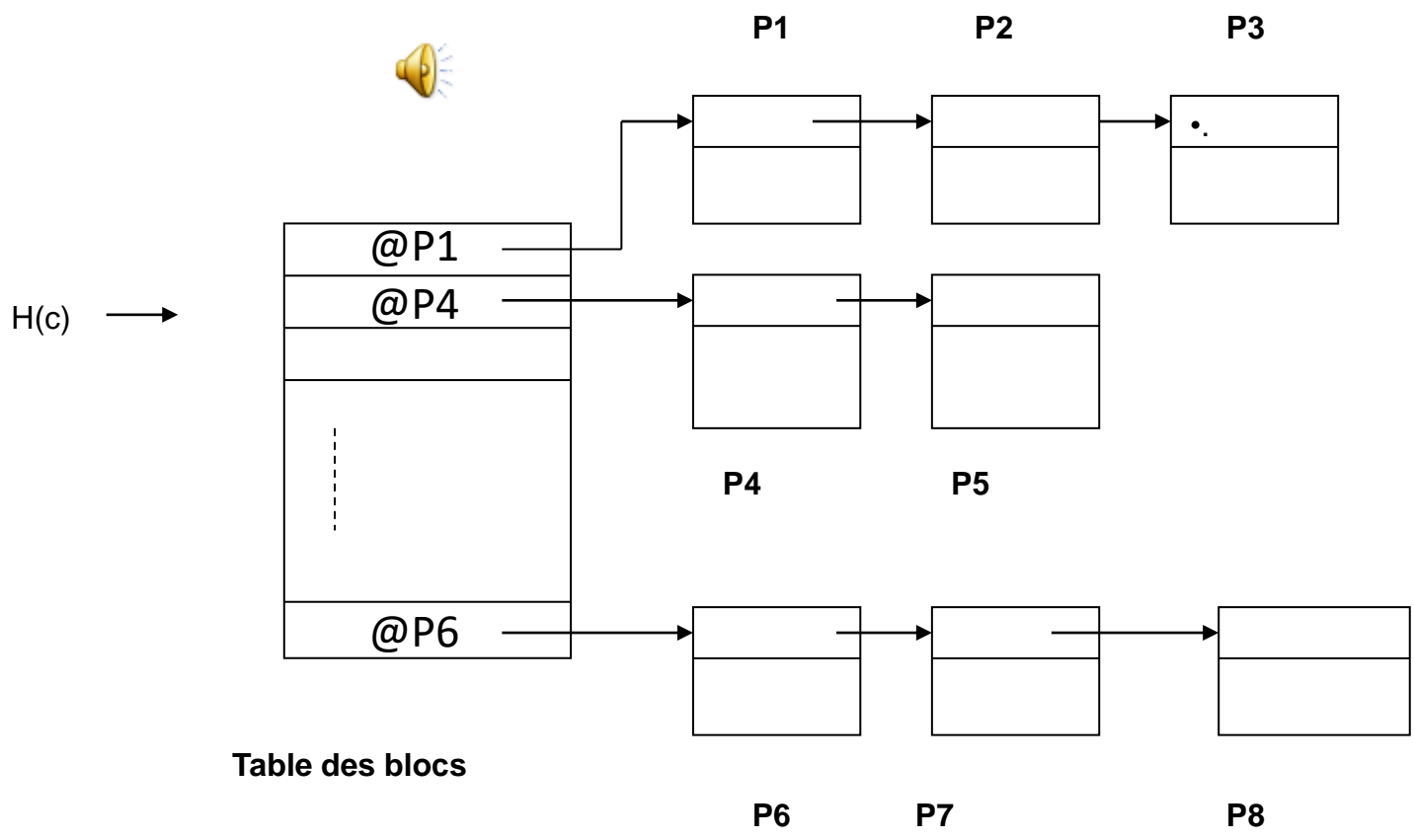


# Algorithme de recherche

entrée : valeur de clé  $c$

- Calcul  $h(c)$  : numéro de bloc
- Consultation de la table des blocs : récupération de la première page du bloc
- Recherche dans cette page l'enregistrement ayant pour clé  $c$ .

.



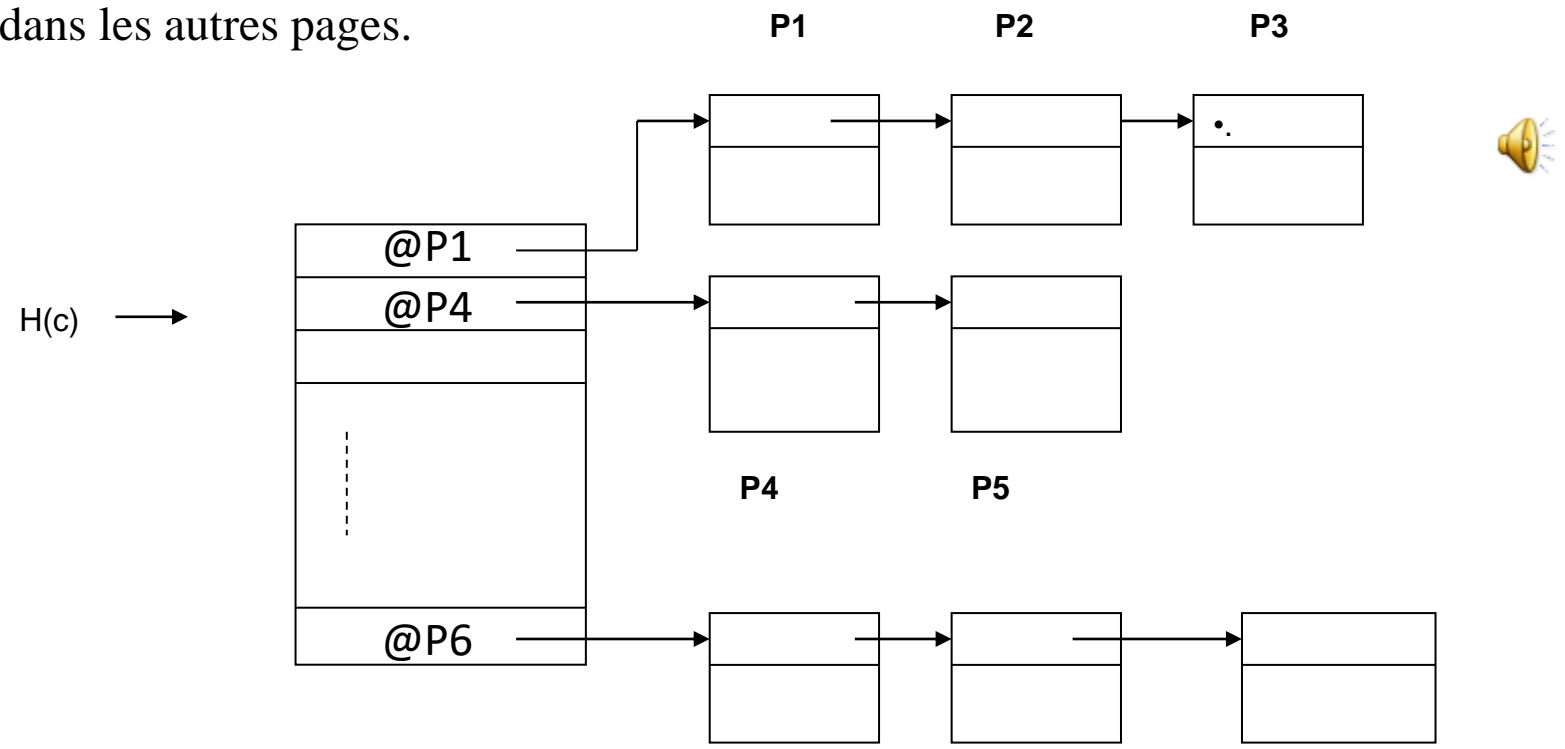
# Algorithme de recherche

entrée : valeur de clé c

- Calcul  $h(c)$  : numéro de bloc
- Consultation de la table des blocs : récupération de la première page du bloc
- Recherche dans cette page l'enregistrement ayant pour clé c.

## Exp:

$H(c)$  est une fonction de hachage  $\text{Mod}[100]$  Donc pour la clé 201. On prend l'entrée N° 2 de la table des blocs. Récupération de l'adresse de la première page du bloc; Accéder à cette page. Chercher dans cette page. Si on ne trouve pas la clé, suivre le lien de chainage pour chercher dans les autres pages.



## Algorithme de recherche

entrée : valeur de clé  $c$

- Calcul  $h(c)$  : numéro de bloc
- Consultation de la table des blocs : récupération de la première page du bloc
- Recherche dans cette page l'enregistrement ayant pour clé  $c$ .

## Algorithme de modification :

- Rechercher l'enregistrement à l'aide de l'algorithme précédent
- Réaliser la modification



## Algorithme de recherche

entrée : valeur de clé  $c$

- Calcul  $h(c)$  : numéro de bloc
- Consultation de la table des blocs : récupération de la première page du bloc
- Recherche dans cette page l'enregistrement ayant pour clé  $c$ .

## Algorithme de modification :

- Rechercher l'enregistrement à l'aide de l'algorithme précédent
- Réaliser la modification



## Algorithme d'insertion :

- Rechercher si le nouvel enregistrement n'existe pas.
- Si non : si le bloc n'est pas saturé alors insérer le nouvel enregistrement, sinon, allouer une nouvelle page, insérer le nouvel enregistrement et chaîner la nouvelle page aux autres.

\*Le bloc est saturé signifie qu'il va y avoir débordement. C'est la gestion des débordements qui va dégrader les performances dans les techniques de hachage.

## Algorithme de recherche

entrée : valeur de clé  $c$

- Calcul  $h(c)$  : numéro de bloc
- Consultation de la table des blocs : récupération de la première page du bloc
- Recherche dans cette page l'enregistrement ayant pour clé  $c$ .

## Algorithme de modification :

- Rechercher l'enregistrement à l'aide de l'algorithme précédent
- Réaliser la modification



## Algorithme d'insertion :

- Rechercher si le nouvel enregistrement n'existe pas.
- Si non : si le bloc n'est pas saturé alors insérer le nouvel enregistrement, sinon, allouer une nouvelle page, insérer le nouvel enregistrement et chaîner la nouvelle page aux autres.

\*Le bloc est saturé signifie qu'il va y avoir débordement. C'est la gestion des débordements qui va dégrader les performances dans les techniques de hachage.

## Algorithme de suppression :

- Rechercher l'enregistrement à supprimer et
- Soit libérer la place qu'occupait cet enregistrement en mettant à jour le chaînage,
- Soit mettre un indicateur de suppression dans l'en-tête de l'enregistrement à supprimer.

# Rappels sur les méthodes classiques de stockage

## 3. Fichiers indexés



### Principe:


Un fichier index contient un ensemble de couples  $(c,p)$  où  $c$  est la clé du premier (des fois la plus grande valeur) enregistrement de la page  $p$ .

- **Index dense**
- **Index non dense**

# Rappels sur les méthodes classiques de stockage

## 3. Fichiers indexés

### Principe:


Un fichier index contient un ensemble de couples  $(c,p)$  où  $c$  est la clé du premier (des fois la plus grande valeur) enregistrement de la page  $p$ . 

- **Index dense** : il contient toutes les clés du fichier.
- **Index non dense**

# Rappels sur les méthodes classiques de stockage

## 3. Fichiers indexés

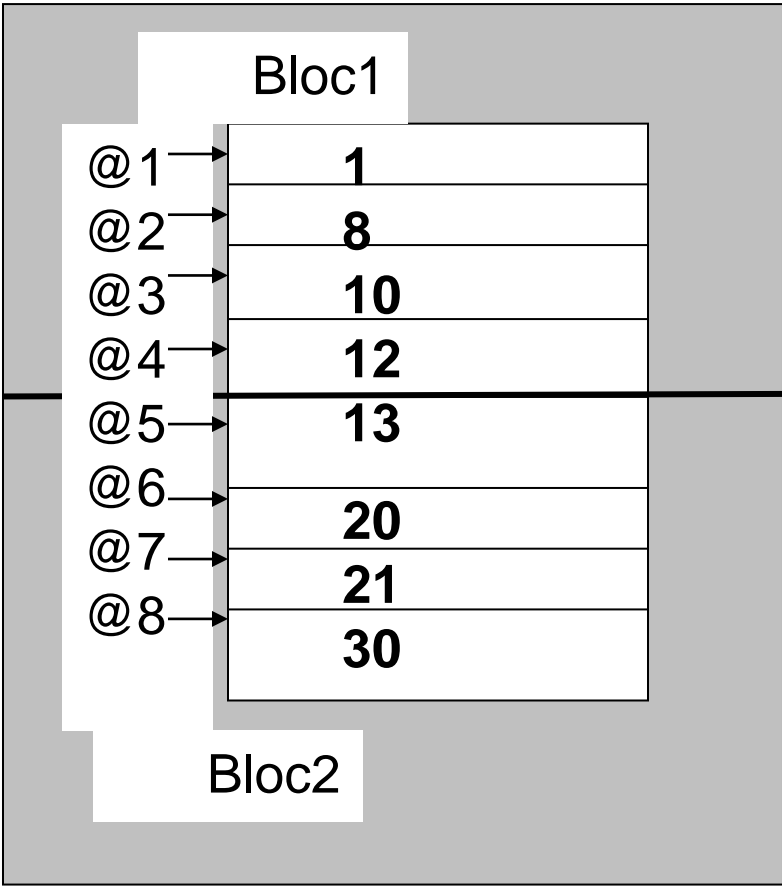
### Principe:

Un fichier index contient un ensemble de couples (c,p) où c est la clé du premier (des fois la plus grande valeur) enregistrement de la page p. 

- **Index dense** : il contient toutes les clés du fichier.
- **Index non dense** : on crée des enregistrements index pour certains enregistrements du fichier : dans ce cas le **fichier est trié** et divisé en blocs. A chaque bloc lui est associée une entrée dans l'index.  
(c,p) = < plus grande clé du bloc, adresse relative du bloc >



Exemple  
Soit le fichier suivant :



***Index dense***

<b>1</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>13</b>	<b>20</b>	<b>21</b>	<b>30</b>
@1	@2	@3	@4	@5	@6	@7	@8



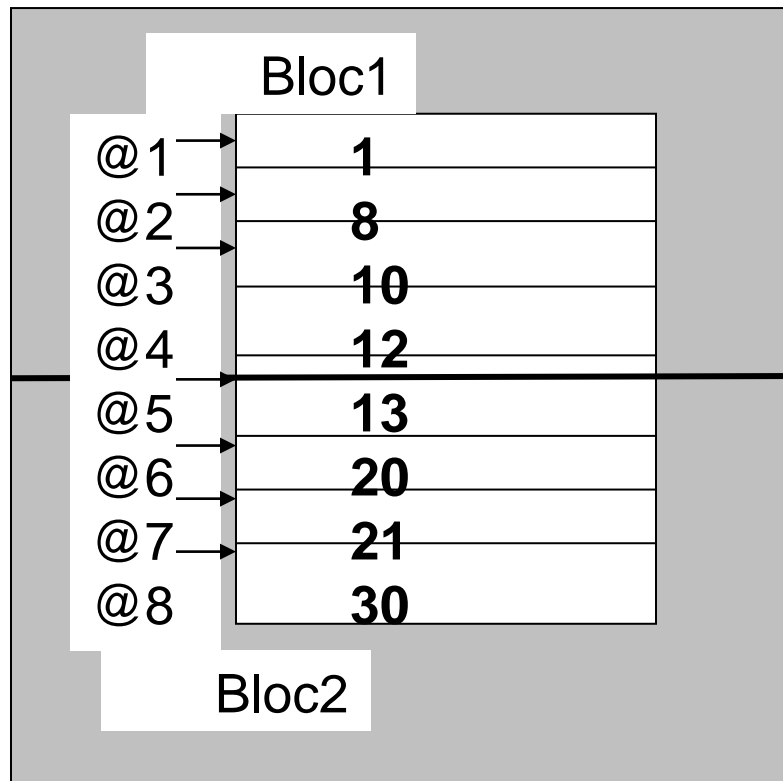
***Index non dense***

(c,p) = < plus grande clé du bloc,  
adresse relative du bloc>

12	30	← Clés
@1	@5	← Adresses de blocs

# 1. Algorithme de recherche :

- Accès à l'index,
- Recherche dans l'index de la clé d'enregistrement désiré,
- Récupération dans l'index de l'adresse relative de l'enregistrement ( si index dense), ou de l'adresse relative du bloc qui le contient (si index non dense),
- Conversion de l'adresse relative en adresse réelle,
- Accès à l'enregistrement ou au bloc,
- Transfert de l'enregistrement dans la zone du programme utilisateur.



## ***Index dense***

1	8	10	12	13	20	21	30
@1	@2	@3	@4	@5	@6	@7	@8

## ***Index non dense***

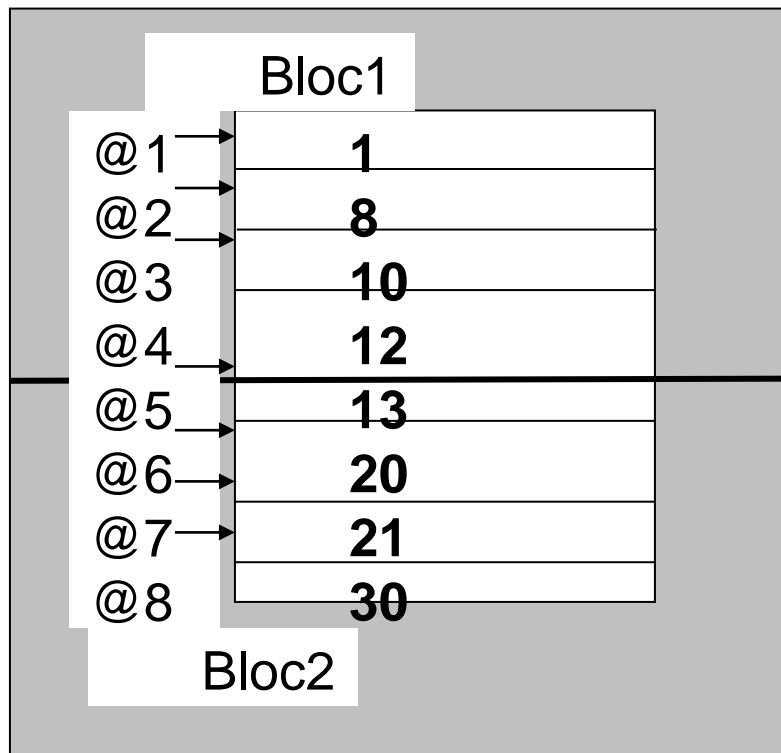


12	30	← Clés
@1	@5	← Adresses de blocs

## 2. Algorithme d'insertion :



- Accès à l'index,
- Détermination de l'emplacement de la page qui doit contenir l'enregistrement, puis détermination de la place de l'enregistrement dans la page.
- Si la place existe (page non saturée), alors insérer l'enregistrement en déplaçant les autres si nécessaire.
- Si la page est pleine, il existe différentes stratégies, entre autres, aller à la page suivante ou allouer une nouvelle page, tout en mettant à jour l'index.



### ***Index dense***

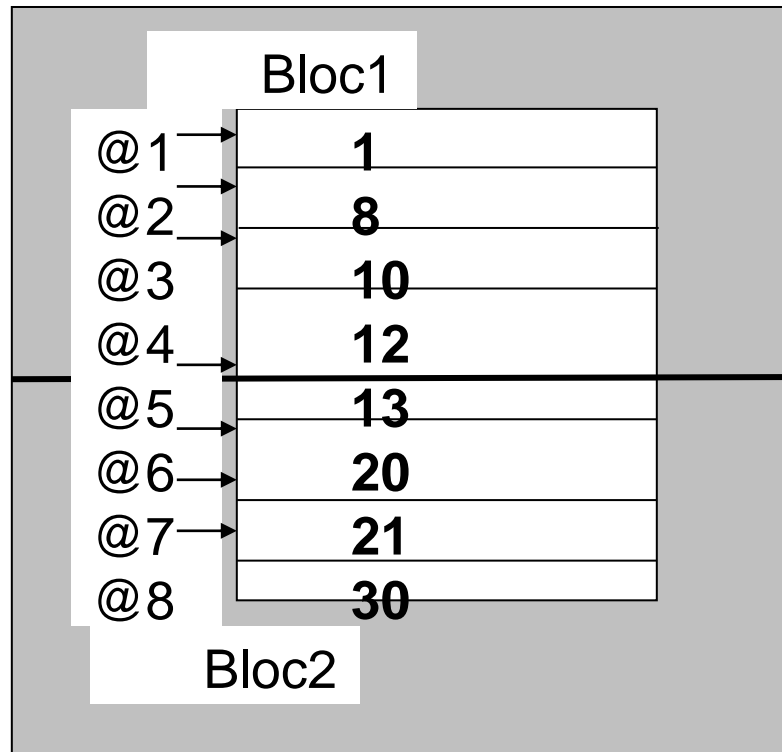
1	8	10	12	13	20	21	30
@1	@2	@3	@4	@5	@6	@7	@8

### ***Index non dense***

12	30	← Clés
@01	@05	← Adresses de blocs

### 3. Algorithme de suppression :

- Appliquer l'algorithme de recherche pour trouver l'enregistrement,
- Soit supprimer réellement l'enregistrement en mettant à jour l'index,
- Soit faire une suppression logique.
- Cas particuliers : si l'enregistrement à supprimer est cité dans l'index, alors une modification de l'index est nécessaire. Lorsqu'une page devient complètement vide, il faut la rendre au système et mettre à jour l'index.



#### *Index dense*

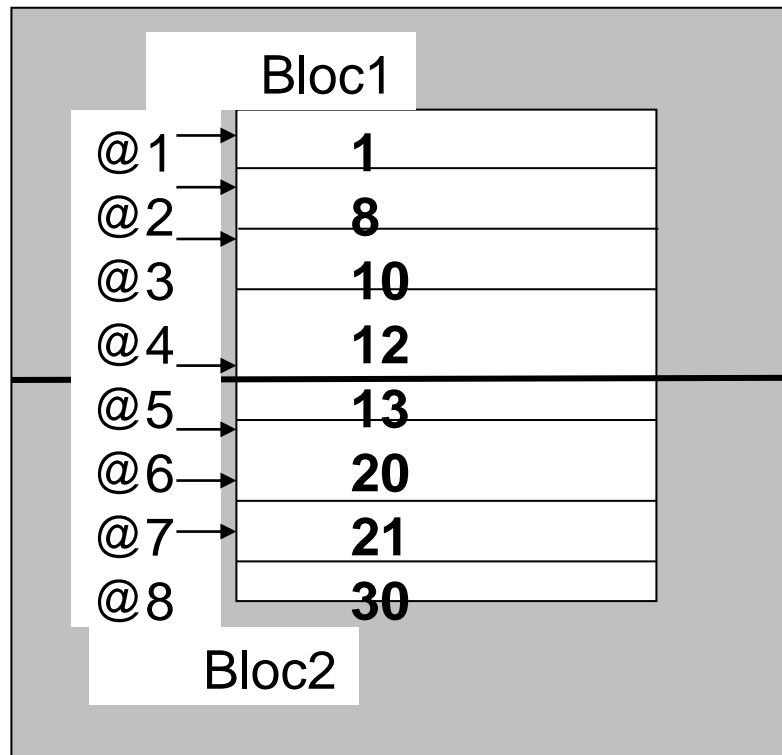
1	8	10	12	13	20	21	30
@1	@2	@3	@4	@5	@6	@7	@8

#### *Index non dense*

<del>12</del> 11	30	← Clés
@01	@05	← Adresses de blocs

#### 4. Algorithme de modification :

- Appliquer l'algorithme de recherche pour trouver l'enregistrement à modifier,
- Réaliser la modification.
- Cas particulier : si la modification porte sur la clé, alors la traiter comme une suppression, suivie d'une insertion.



#### ***Index dense***

<b>1</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>13</b>	<b>20</b>	<b>21</b>	<b>30</b>
@1	@2	@3	@4	@5	@6	@7	@8

#### ***Index non dense***

12	30	← Clés
@01	@05	← Adresses de blocs



Un index étant lui-même un fichier, il n'y a aucune raison, si celui-ci est volumineux, de définir un autre niveau d'index et ainsi de suite : nous obtenons alors un **index hiérarchisé à plusieurs niveaux** ( exemple la méthode séquentielle indexée ISAM dans laquelle il pouvait y avoir deux ou trois niveaux d'index : index de cylindres, index de pistes, et éventuellement index maître ).