第六章 索引

章成源

湖南大学-信息科学与工程学院-计算机科学系

办公室：院楼403

Email：cyzhangcse@hnu.edu.cn

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

6.1 顺序表索引

- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

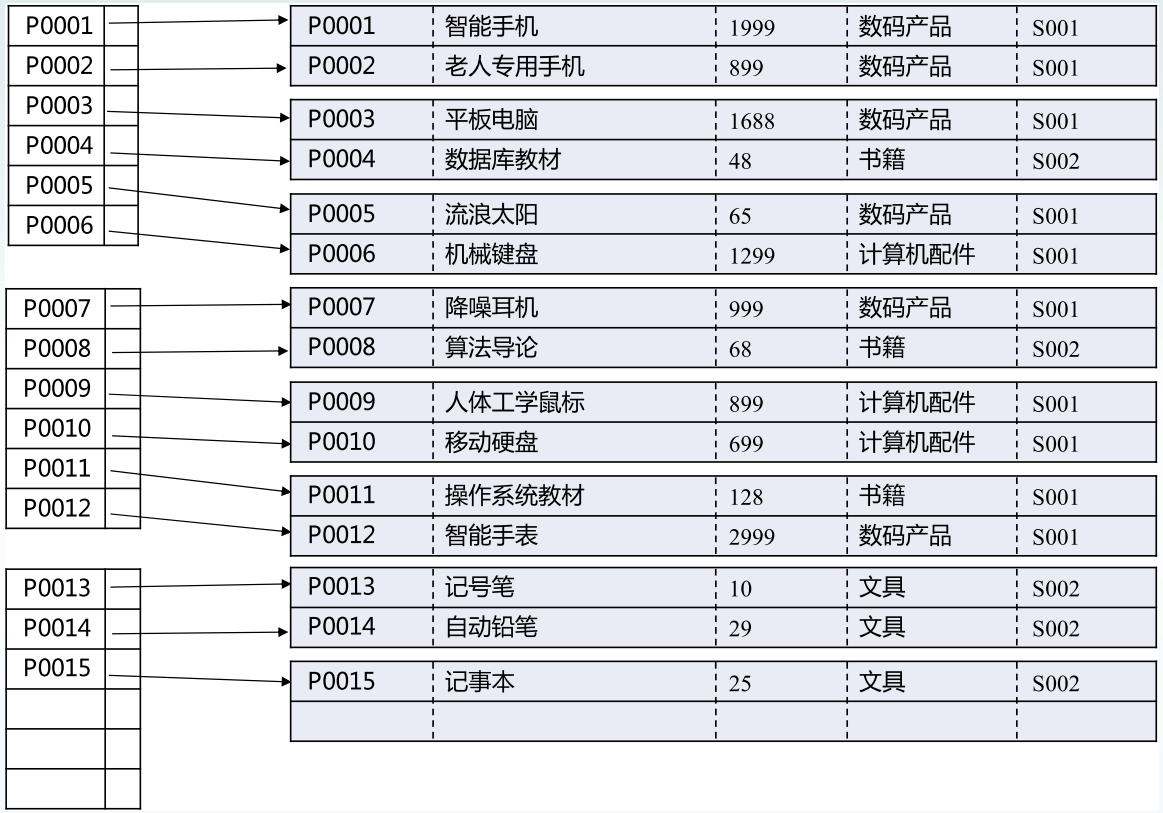
6.1 顺序表索引

- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

稠密索引

- 什么是稠密索引
  - 索引块中存放每条记录的码以及指向记录本身的指针
  - 示例

稠密索引



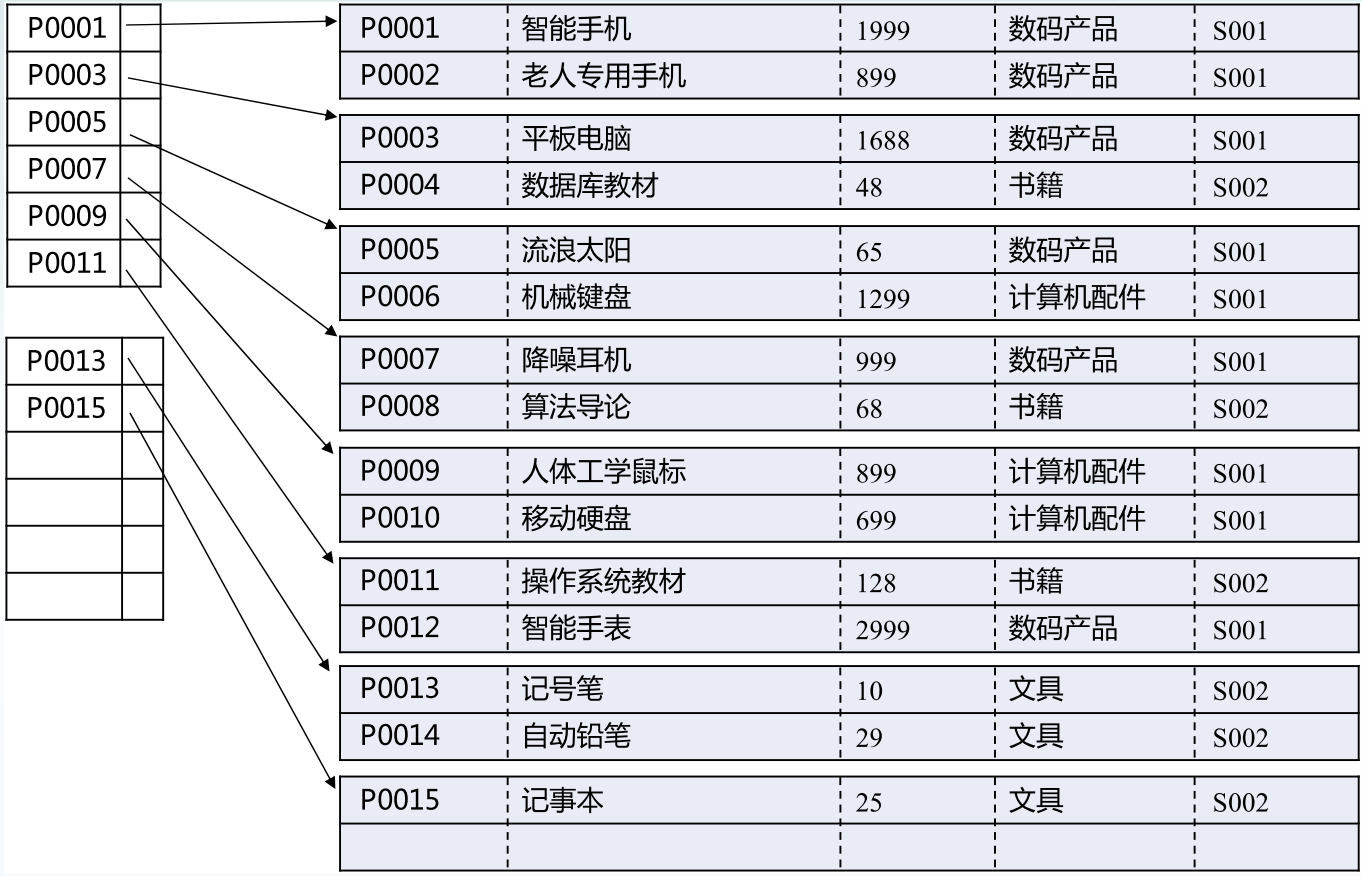
顺序文件

顺序表索引

- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

稀疏索引

- 什么是稀疏索引
  - 索引块中存放每个数据块中第一条记录的码值及指向该记录的指针。
  - 稀疏：每个存储块只有一个索引项。
  - 示例



稀疏索引

- **稀疏索引的查找方法（查找码为K的记录）**
  - 首先用二分查找法查找码值小于或等于K的最大码值
  - 然后根据它的指针找到相应的数据块
  - 在该数据块中搜索，查找码值为K的记录

6.1 顺序表索引

- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

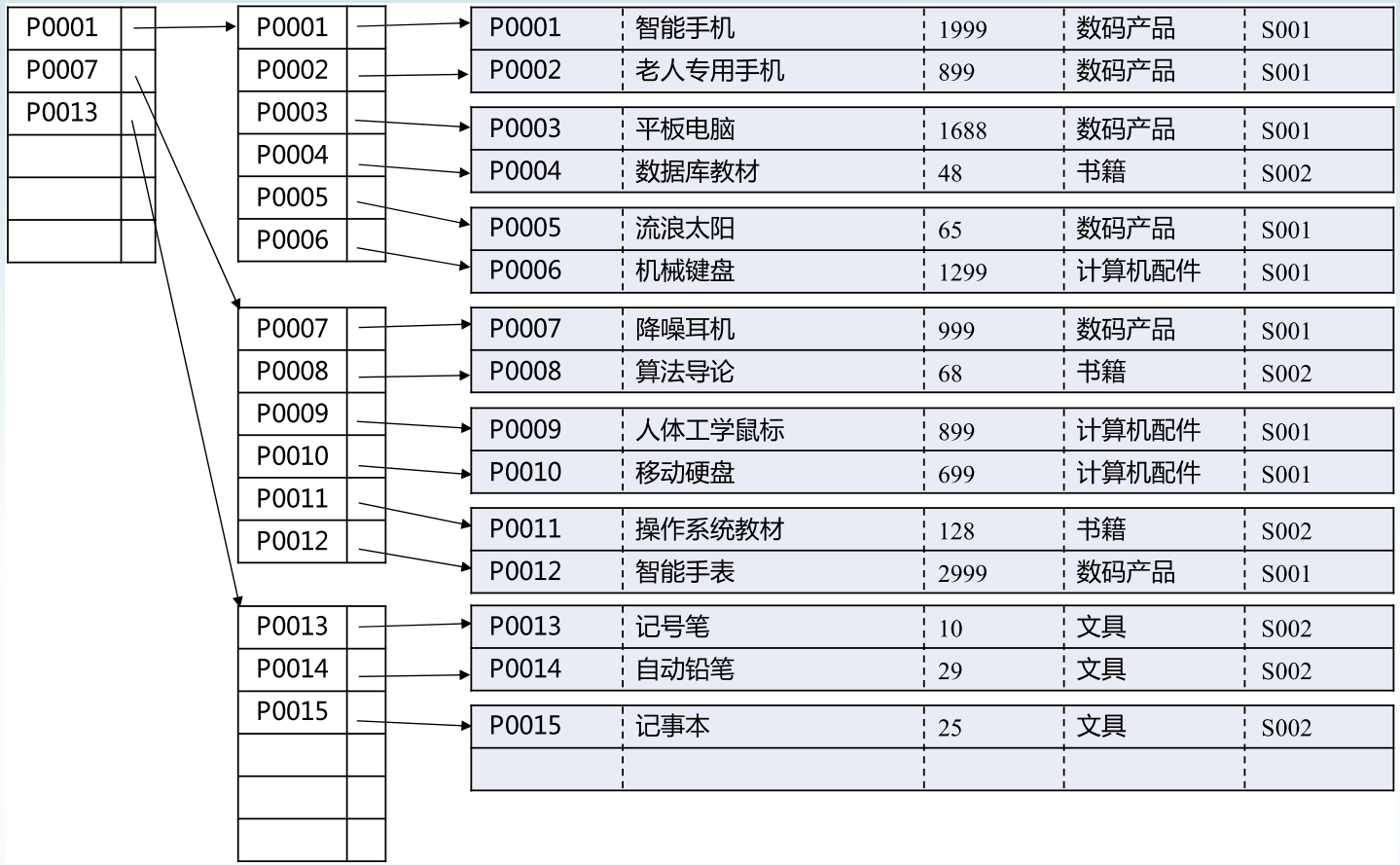
多级索引

- **多级索引的引入**
  - 索引较小时，可以将其存放到内存或磁盘的预留存储区中，从而使索引查找速度较快。当索引较大时，可以在索引上再建一级索引，并将新的索引本身存放到某个固定的地方，这就是多级索引。
  - 通过在索引上再建索引，能够使第一级索引更为有效。
  - 二级索引或更高级的索引必须是稀疏索引。

多级索引

- 示例

稀疏二级索引　　顺序文件



多级索引

- **多级索引的查找方法**

  (查找码为K的记录，以两级索引为例)

  - 首先用二分查找法查找二级索引，找到码值小于或等于K的最大码值
  - 根据该最大码值对应的指针找到一级索引的相应存储块B
  - 若存储块B不在内存，将其读入内存
  - 利用一级索引查找码值为K的记录
    - 如果一级索引是稠密索引：同稠密索引查找方法
    - 如果一级索引是稀疏索引：同稀疏索引查找方法

6.1 顺序表索引

- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

6.1 顺序表索引

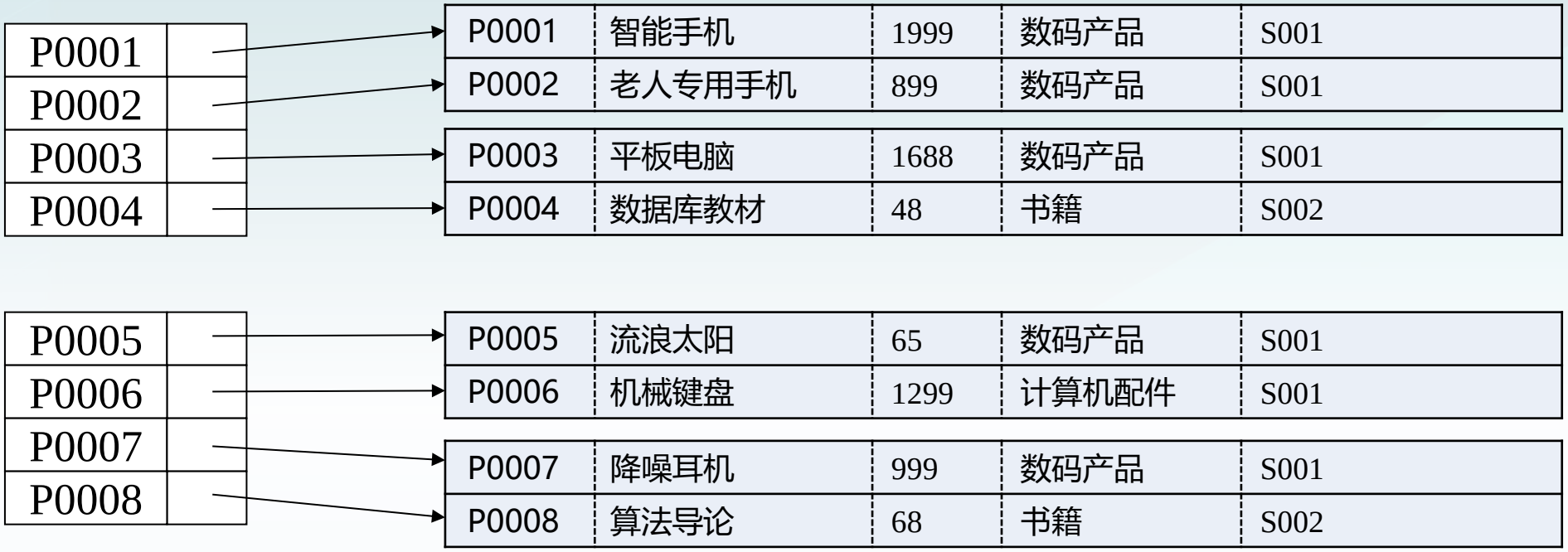
- 6.1.1 稠密索引
- 6.1.2 稀疏索引
- 6.1.3 多级索引
- 6.1.4 索引维护

索引维护

- **删除**
  - 从稠密索引中删除某个记录
  - 从稀疏索引中删除某个记录
- **插入**
  - 向稀疏索引中插入某个记录
  - 向稠密索引中插入某个记录

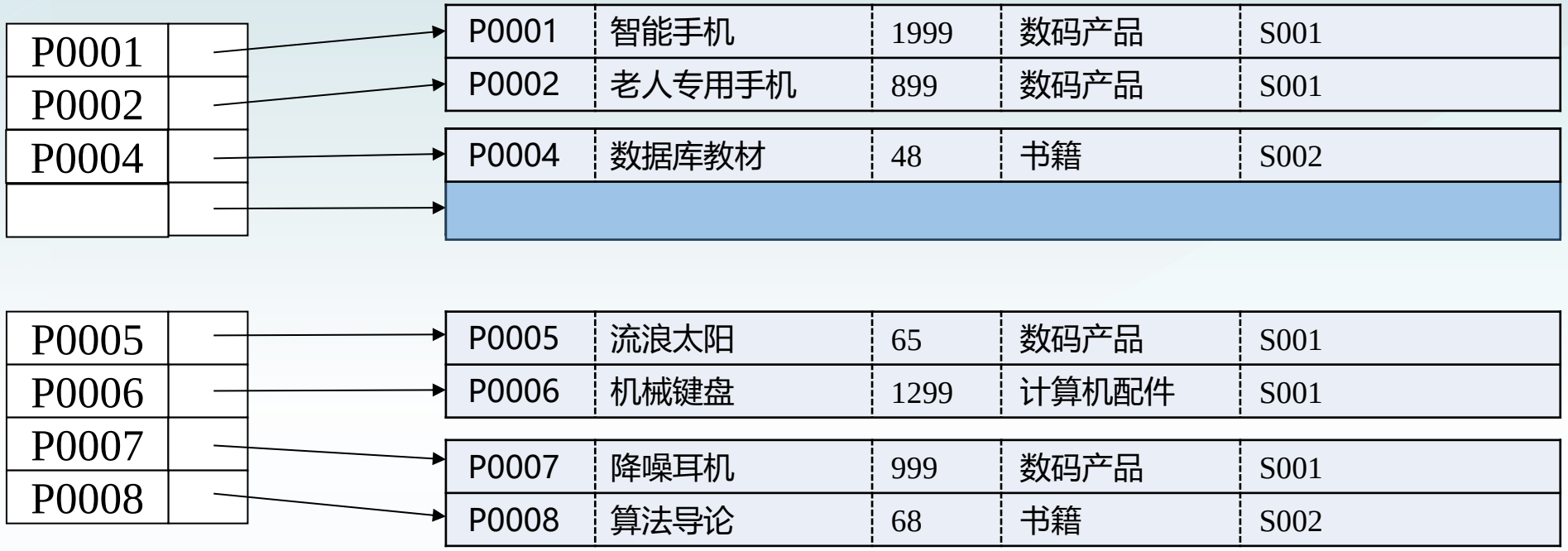
索引维护

- 从稠密索引中删除某个记录



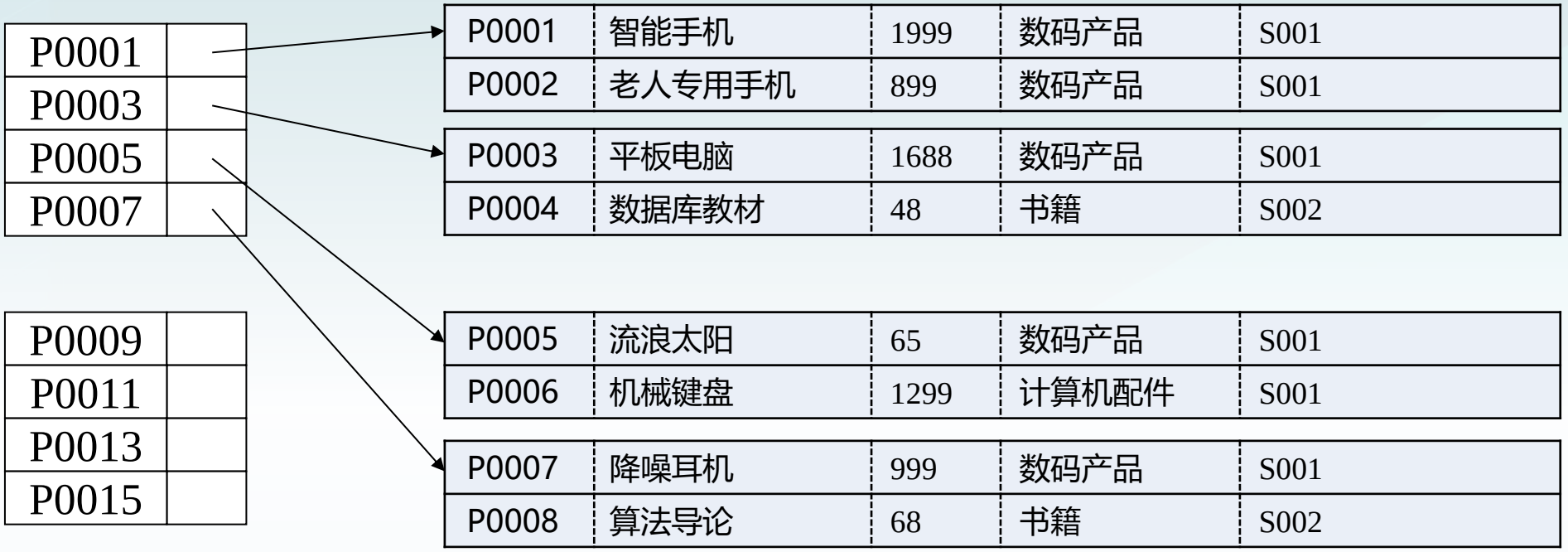
索引维护

- 从稠密索引中删除某个记录
  - 删除 记录 P0003



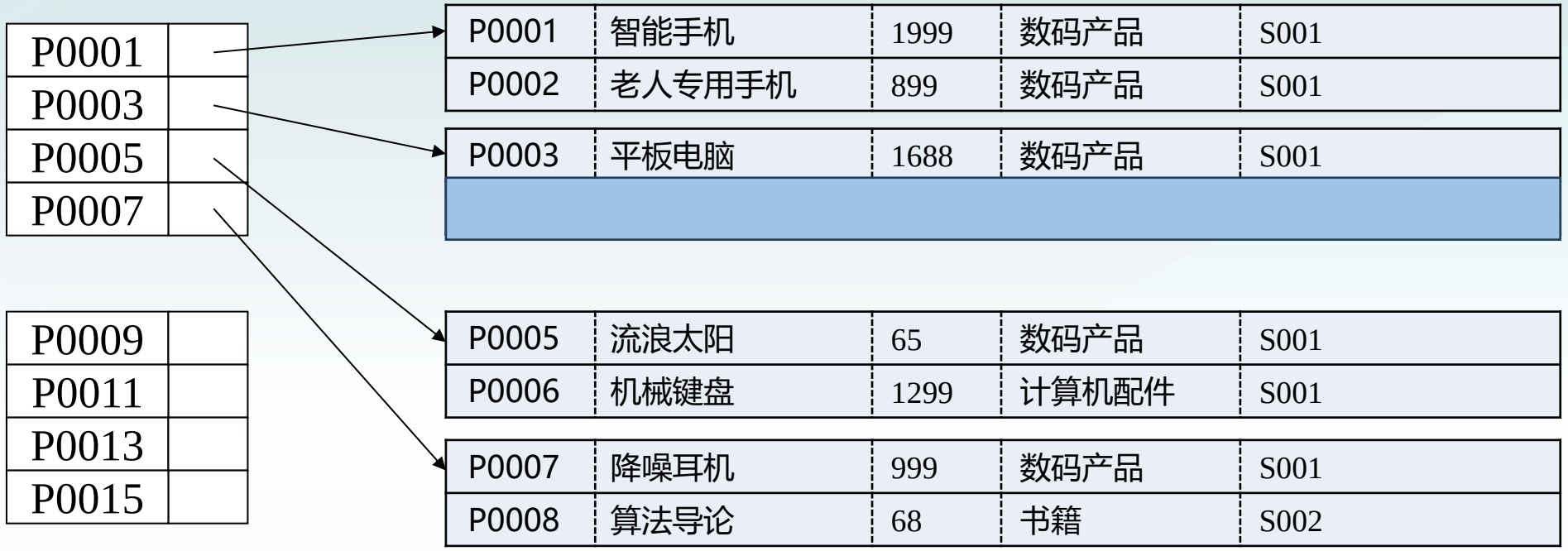
索引维护

- 从稀疏索引中删除某个记录



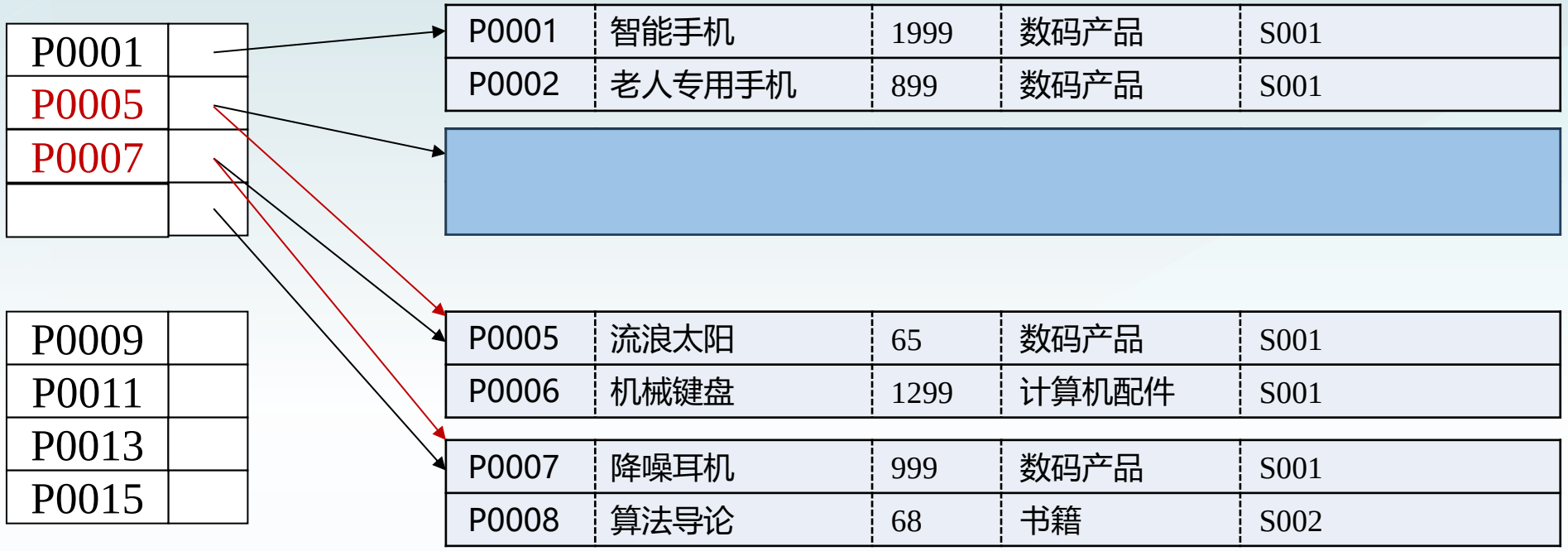
索引维护

- 从稀疏索引中删除某个记录
  - 删除 记录 P0004



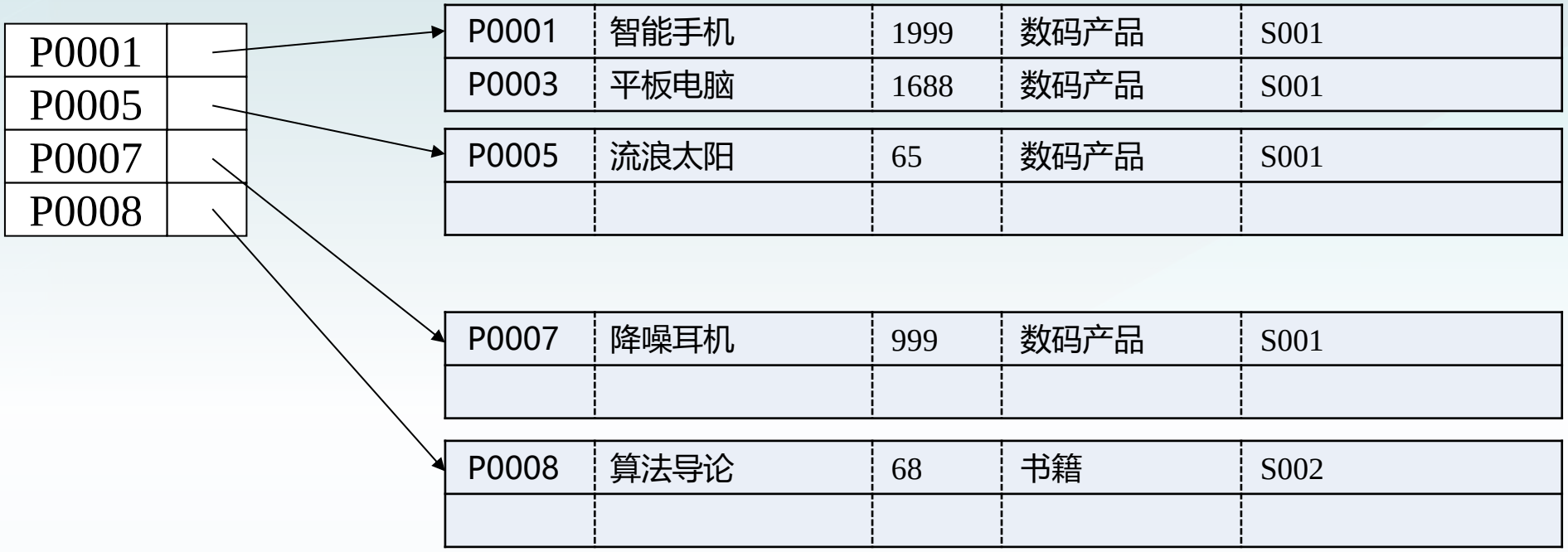
索引维护

- 从稀疏索引中删除某个记录
  - 删除 记录 P0003和P0004



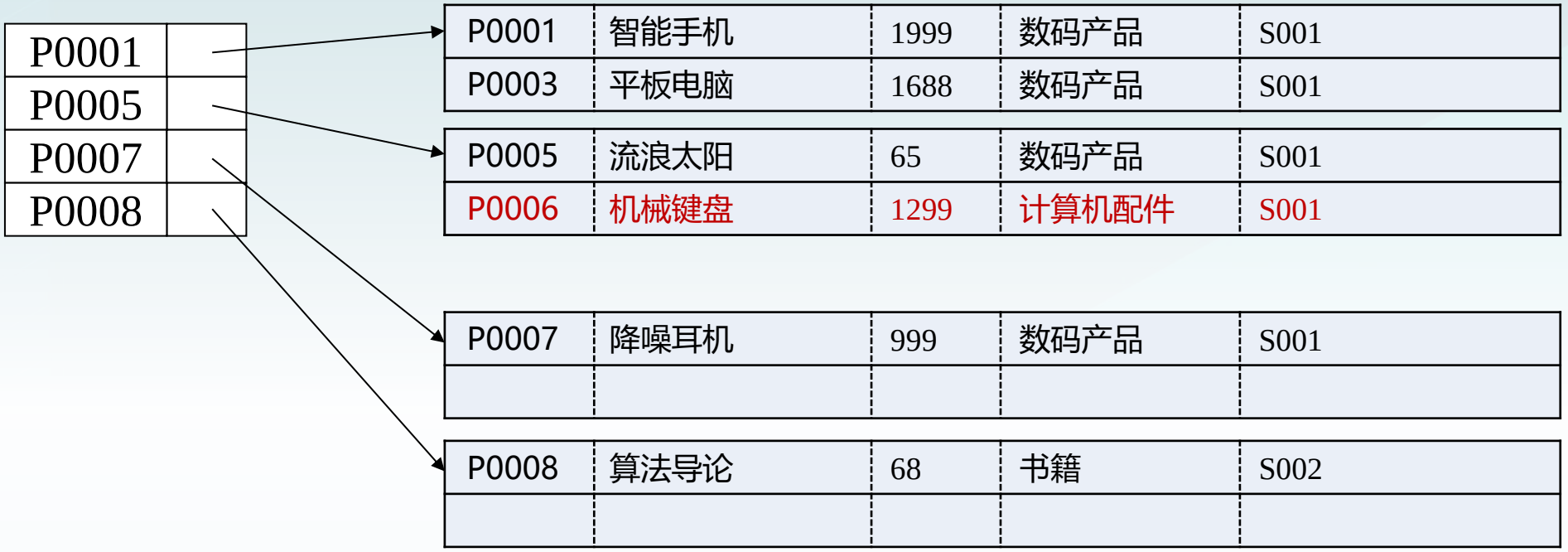
索引维护

- 向稀疏索引中插入某个记录



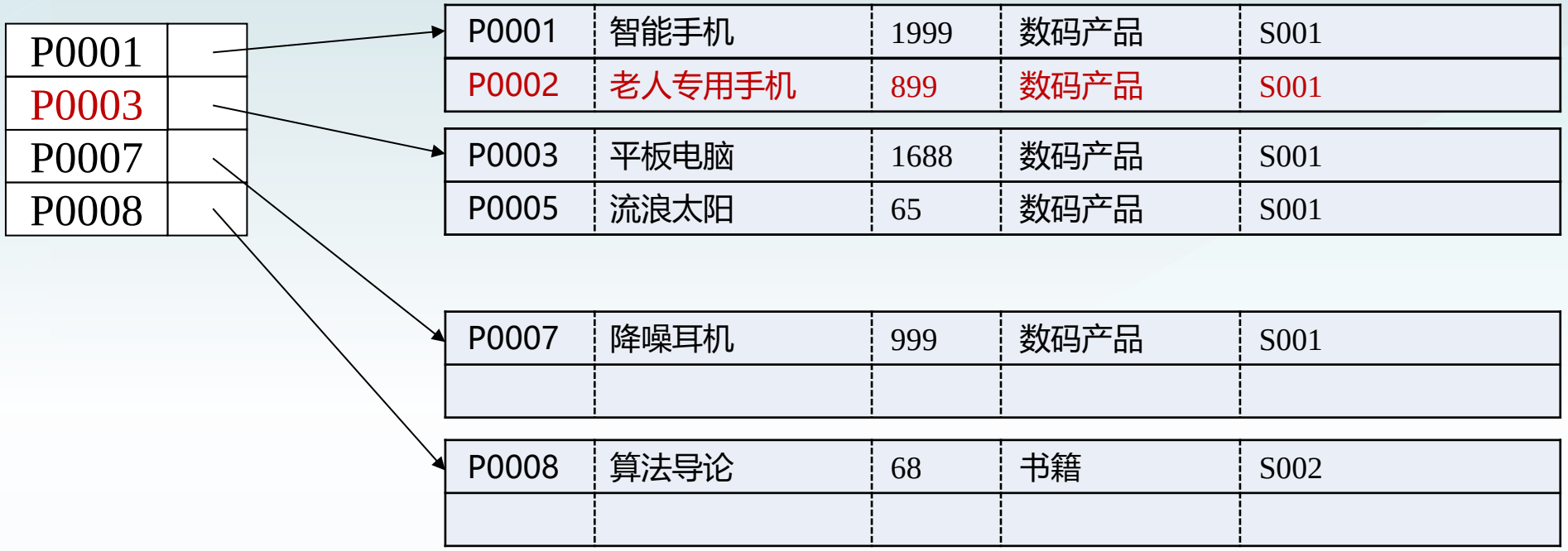
索引维护

- 向稀疏索引中插入某个记录
  - 插入 记录 P0006



索引维护

- 向稀疏索引中插入某个记录
  - 插入 记录 P0002



索引维护

- 向稀疏索引中插入某个记录
  - 插入 记录 P0004

索引维护

- **向稠密索引中插入记录**
  - 类似于稀疏索引的插入方式
  - 代价更为昂贵

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

6.2 辅助索引

- 6.2.1 辅助索引的特点
- 6.2.2 辅助索引的实现技术

6.2 辅助索引

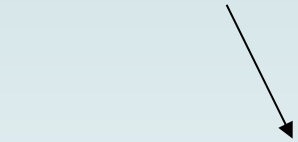
- 6.2.1 辅助索引的特点
- 6.2.2 辅助索引的实现技术

辅助索引的特点

- 辅助索引不决定数据文件中记录的存放位置，只能指明记录的当前存放位置。
- 辅助索引总是稠密索引。原因：辅助索引不影响记录的存储位置，因而不能根据它来预测码值不在索引中显式指明的任何记录位置。
- 建辅助索引的属性通常取重复值

辅助索引的特点

非排序属性



P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002

稀疏索引不适用

辅助索引的特点

Category属性稠密索引

P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002

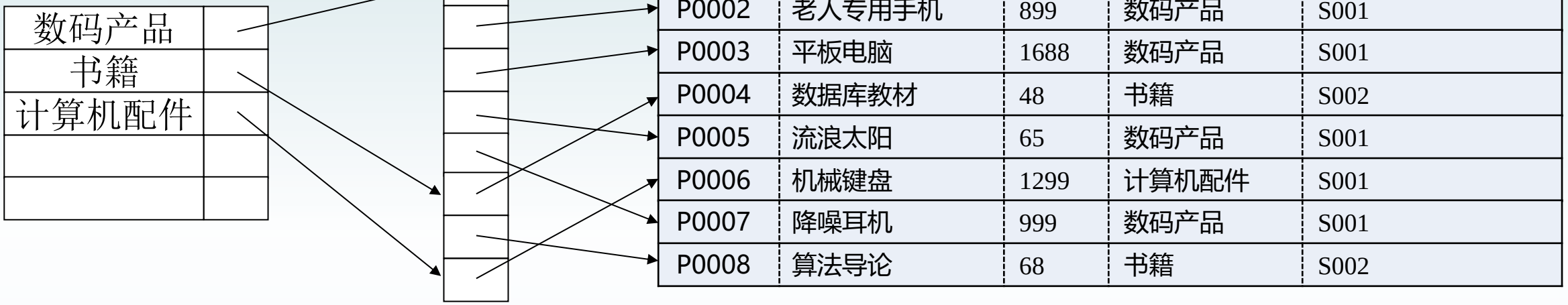
- 建辅助索引的属性往往会取重复值。
- 去掉重复取值的索引项，可以减小索引查找的开销。

6.2 辅助索引

- 6.2.1 辅助索引的特点
- 6.2.2 辅助索引的实现技术

辅助索引的特点

去掉重复索引项的辅助索引—利用指针桶



P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 **B+树索引**
- 6.4 哈希索引
- 6.5 **Bitmap索引**
- 6.6 小结

第6章 B+树索引

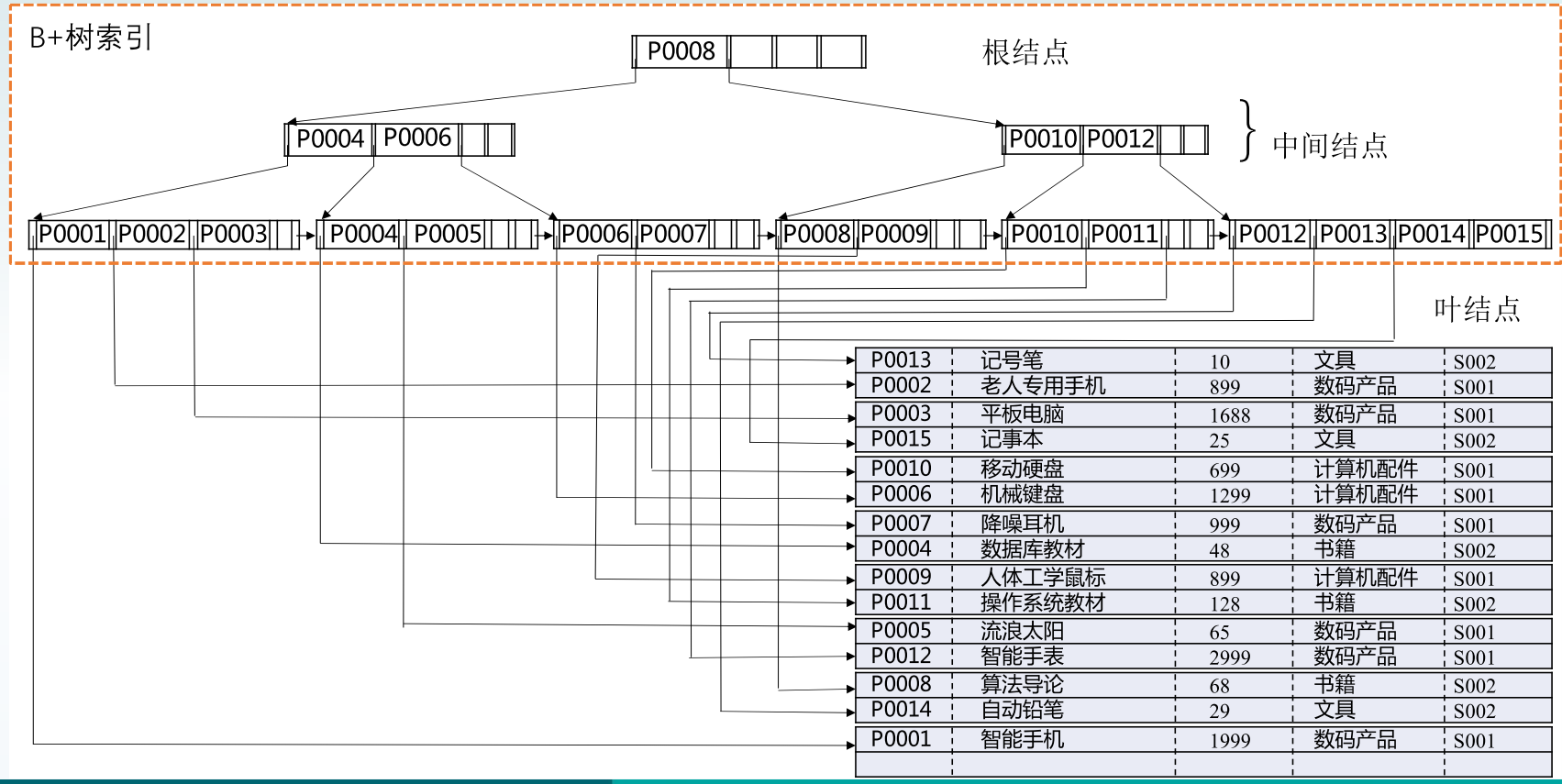
- 6.3.1 B+树
- 6.3.2 B+树的查找
- 6.3.3 B+树的维护

第6章 B+树索引

- 6.3.1 B+树
- 6.3.2 B+树的查找
- 6.3.3 B+树的维护

B+树

- **什么是B+树**
  - 基本B树是为解决大型索引的组织和维护而产生的一种数据结构。
  - B+树 = B树 + 顺序集
  - 示例，秩n=5

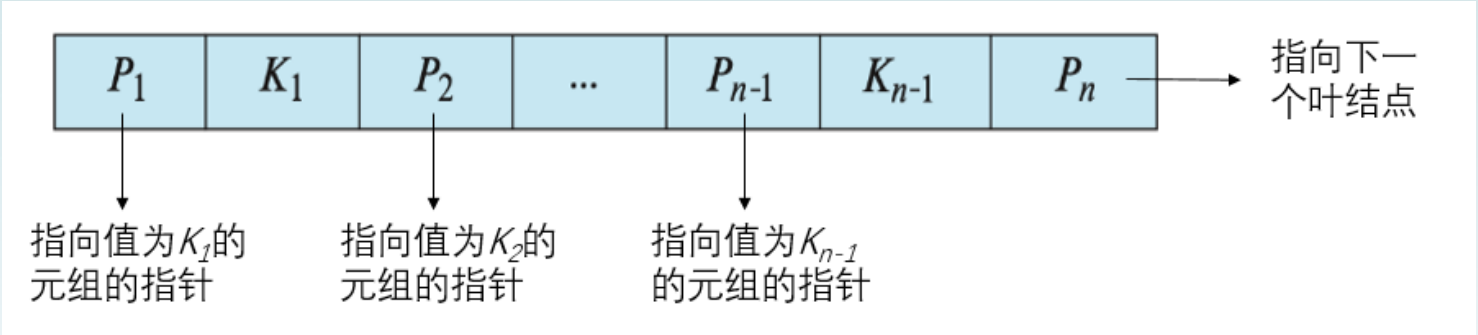


B+树

- **一棵秩(order)为 n的B+树具有下列特征：**
  - 每个结点最多包含n-1个key。
  - 除了根结点外，每个结点最少包含[n-1]/2个key（根结点最少含有一项）。
  - 含有j-1项的非叶结点，有j个指针，分别指向其j个孩子（叶结点除外，它没有孩子）。
  - 所有的叶结点都在同一级上。

B+树

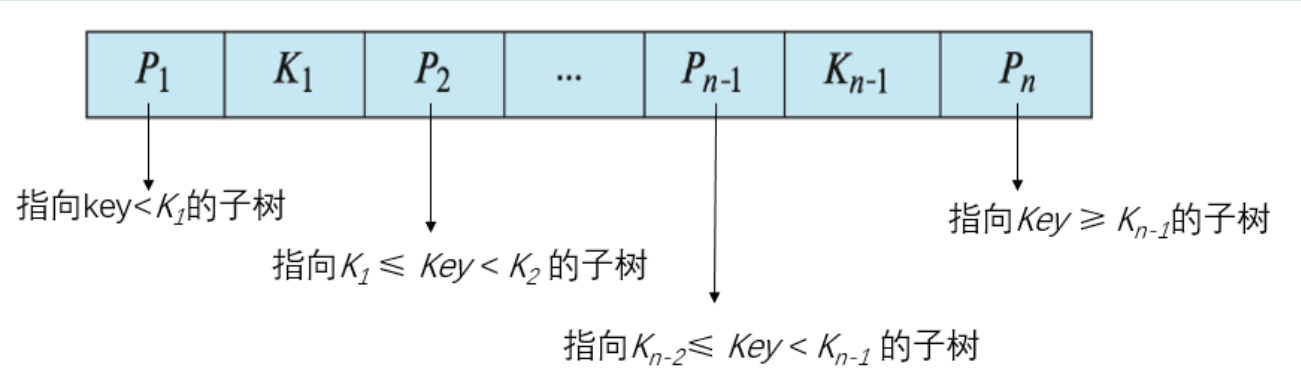
- **B+树索引的叶结点结构**



- $K_i(1 \le i \le j)$是属性值，并且按序排放，即$K_i < K_{i+1}$，P_i是指针。
- 指针P_i指向关系表中属性值为K_i的元组。
- 指针P_n指向其兄弟叶结点，最后一个叶结点的P_n为空。

B+树

- **B^+树索引的非叶结点结构**



  - $K_i(1 \leq i \leq j)$是属性值，并且满足$K_i < K_{i+1}$。
  - 指针P_i指向其下层的孩子结点，P_i所指向的子树中属性值均小于K_i大于等于K_{i-1}。
  - 在实现时一般在结点中附加一项信息指出结点当前所含有的实际项数j。

第6章 B+树索引

- 6.3.1 B+树
- 6.3.2 B+树的查找
- 6.3.3 B+树的维护

B+树的查找

- **点查询**
  - 从根到叶的一条通路。（查找通路）
- **范围查询**
  - 随机查找通路的末端结点为顺序查找提供了入口点，从这个入口点开始沿着顺序集就可以进行顺序查找。
  - 若是从头开始顺序查找，可以从顺序集的链头SH开始。

第6章 B+树索引

- 6.3.1 B+树
- 6.3.2 B+树的查找
- 6.3.3 B+树的维护

B+树的插入

B+树的插入：

(a) 直接插入

(b) 叶溢出

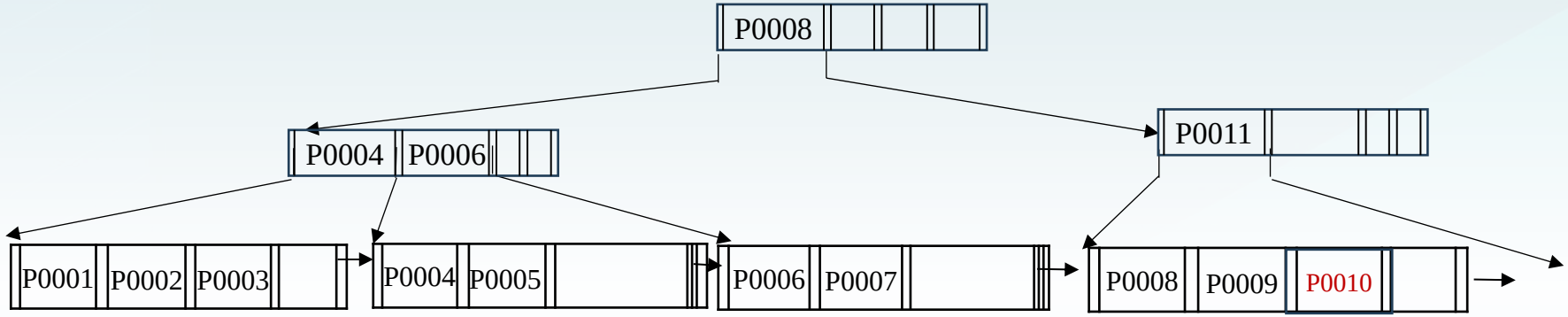
(c) 非叶溢出

(d) 建立新根

B+树的插入

(a) 直接插入 (n=5)

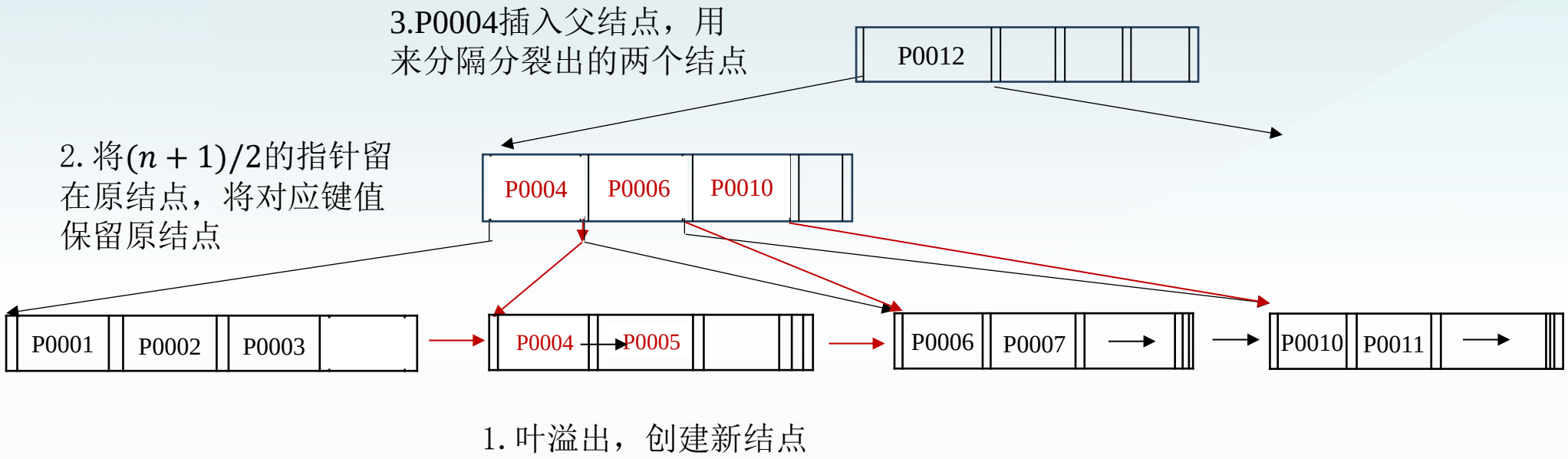
- 插入 key = P0010



B+树的插入

(b) 叶溢出(n=5)

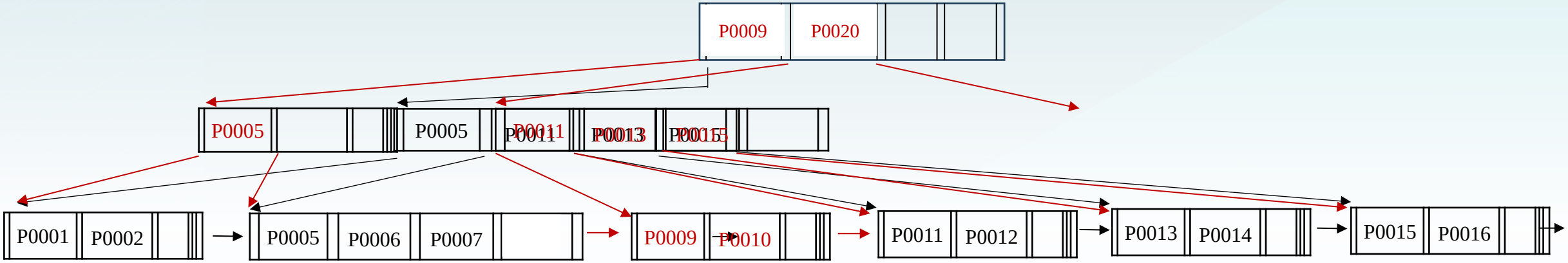
- 插入 key = P0004



B+树的插入

(c) 非叶溢出(n=5)

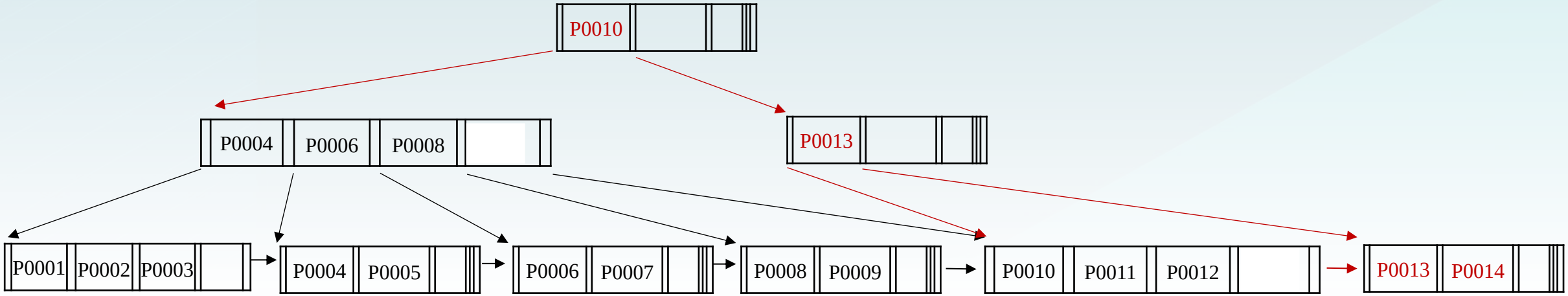
- 插入 key = P0009



B+树的插入

(d) 建立新根(n=5)

- 插入 key = P0014



B+树的删除

B+树的删除：

(a) 直接删除

(b) 与邻居联合

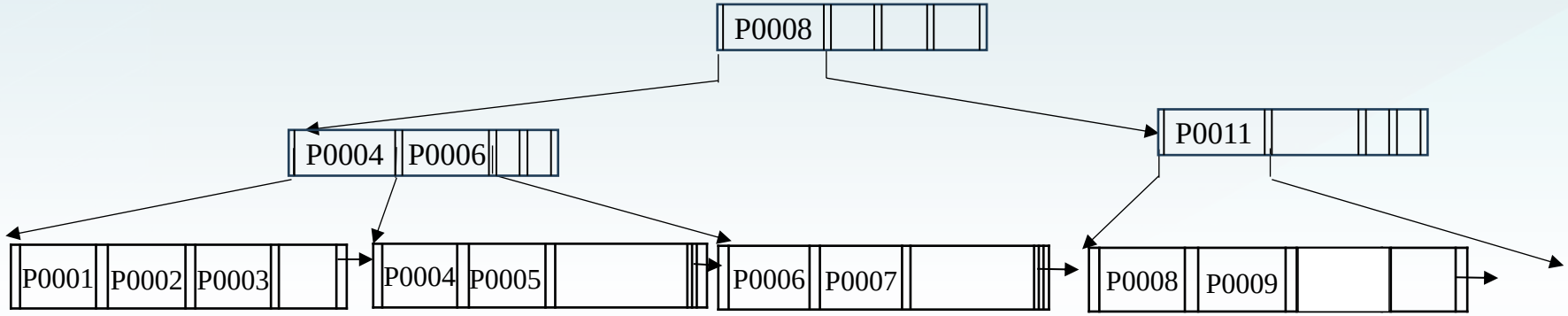
(c) 重新分配键值

(d) 非叶的情况

B+树的删除

(a) 直接删除 (n=5)

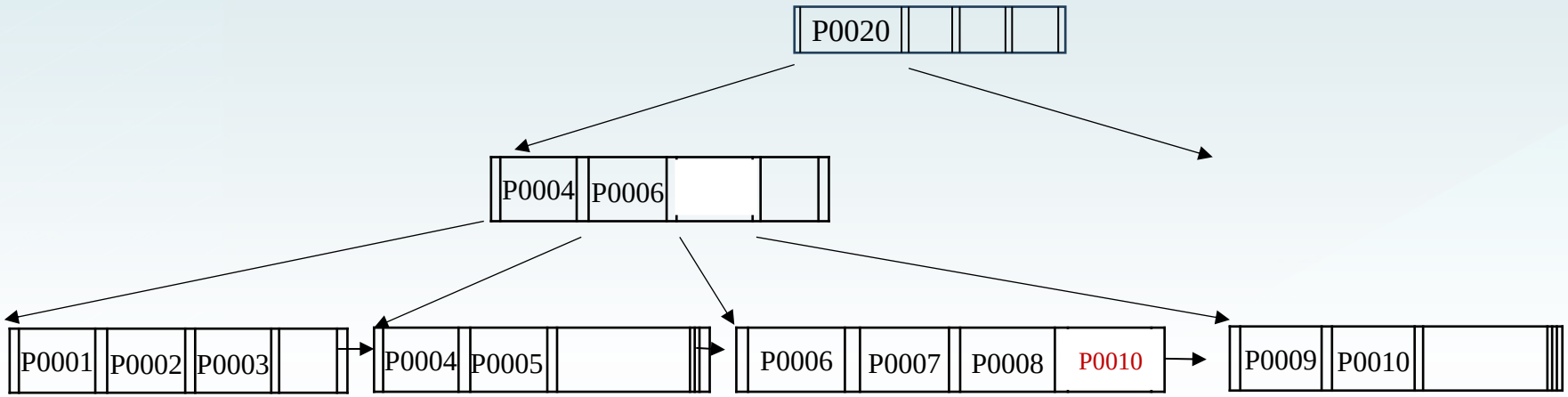
- 删除 key = P0010



B+树的删除

(b) 联合邻居 (n=5)

- 删除 key = P0009

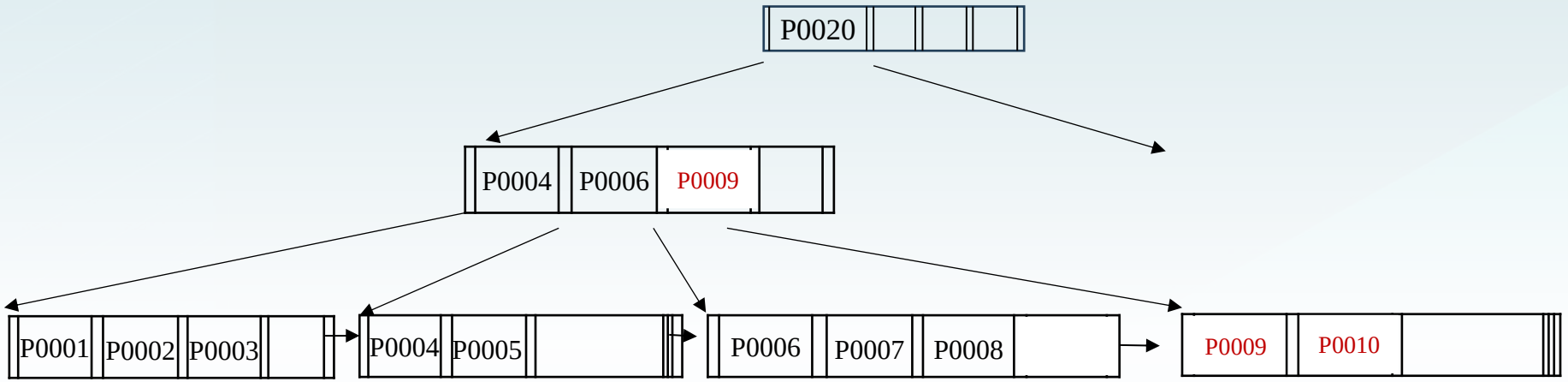


判断左结点是否存在空闲指针，如有则合并

B+树的删除

(c)重新分配键值 (n=5)

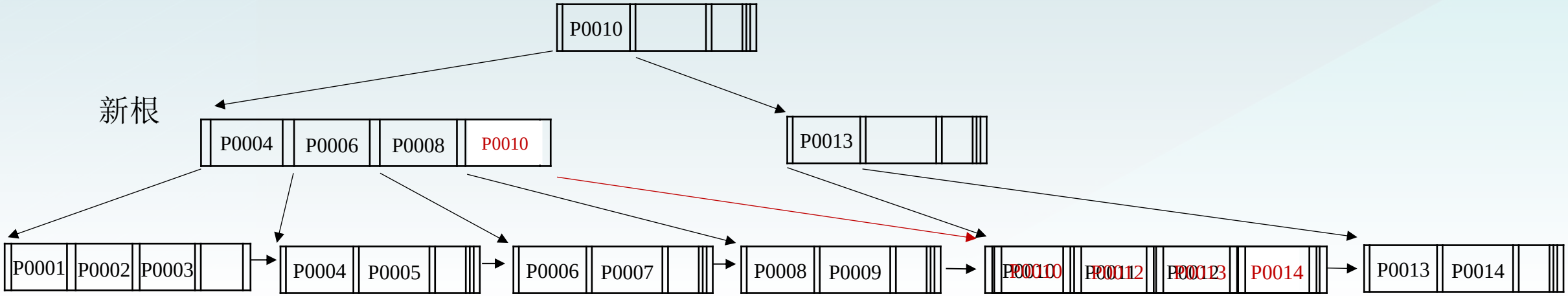
- 删除 key = P0011



B+树的删除

(d) 非叶情况(n=5)

- 删除 key = P0011



第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

第6章 哈希索引

- 6.4.1 静态哈希索引
- 6.4.2 动态哈希索引

第6章 哈希索引

- 6.4.1 静态哈希索引
- 6.4.2 动态哈希索引

哈希函数

哈希函数的两种方式

1. 直接方式

- 把整个值转化为数值型。
- 取模计算相应的哈希值。

哈希函数

2. 间接方式

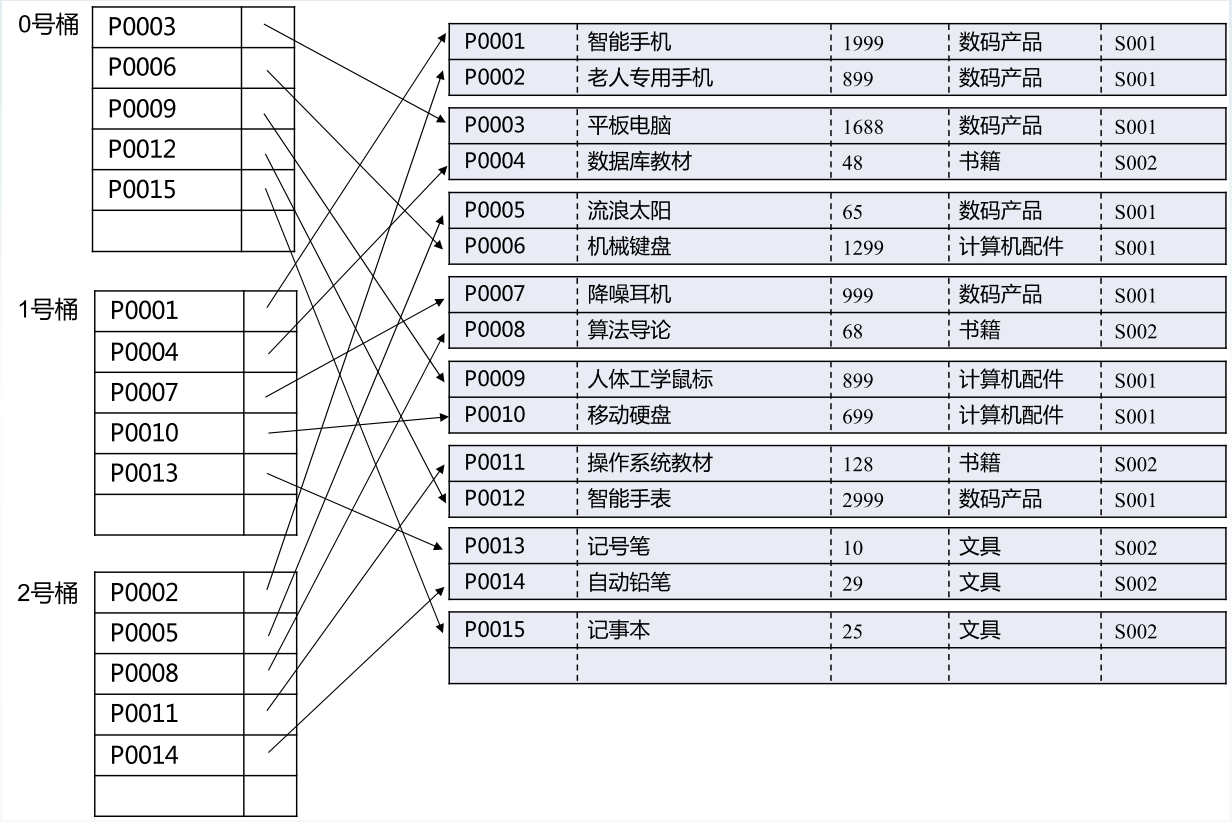
- 把关键字的值化为二进制序列。
- 把此二进制序列划分成定长的若干组，若最后一组数目不足则补0。
- 把若干组二进制迭加起来，得一整数。
- 以存贮区桶的个数除此整数，取其余数作为如上关键字对应的桶号。

哈希表

- **哈希表**
  - 由一组桶组成，一个桶对应一个或多个物理块。
  - 桶中存放被映射到该桶的哈希索引项，每条索引项包括索引属性值和指向相应记录的指针。

哈希表

- 例子
  - Products表*PID*属性上的哈希索引，每个桶最多可以存放6个索引项；
  - 哈希函数是取学号最后2位，将其转换为数值型，除以3并取模作为桶号。



哈希表

- **桶溢出**
  - 哈希桶的空间是有限的，当某个桶的存储空间不足时，会发生桶溢出

- **主要原因：**
  - 哈希桶的数量不足，不能存放所有的索引项。
  - 属性取值存在偏斜，某些属性值过多。
  - 哈希函数设计不合理，无法将索引项均匀地映射到每个桶，导致某个桶的数据过多。

第6章 哈希索引

- 6.4.1 静态哈希索引
- 6.4.2 动态哈希索引

动态哈希索引

背景：

- 随着数据的不断增加，桶溢出就成为静态哈希索引不可回避的问题。

动态哈希索引：

- 随着关系表的增大，逐渐扩大桶的数目，是更适合数据库应用特征的哈希索引结构。
- 动态哈希索引包括可扩展哈希索引（Extensible Hashing）和线性哈希索引（Linear Hashing）两类 。

可扩展哈希索引

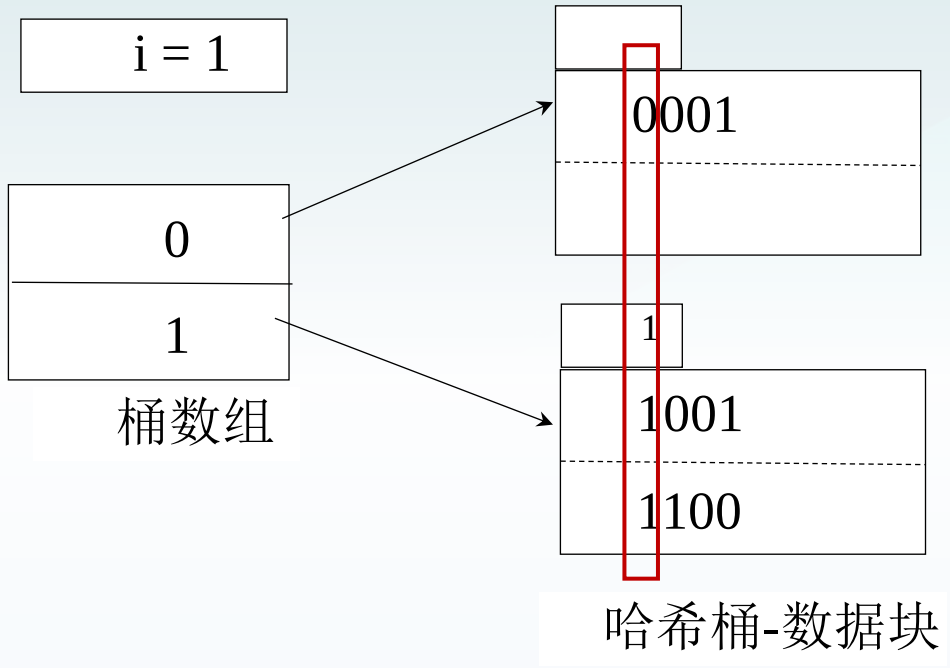
可扩展哈希索引

- 可扩展哈希索引基本结构
- 可扩展哈希索引插入操作
- 可扩展哈希索引删除操作

可扩展哈希索引

可扩展哈希索引基本结构：

- 可扩展哈希表使用一个指针数组作为目录来表示桶，其中的指针指向该桶的数据块。
- 使用最左边的i位哈希值。



可扩展哈希索引插入操作

可扩展哈希索引插入操作：

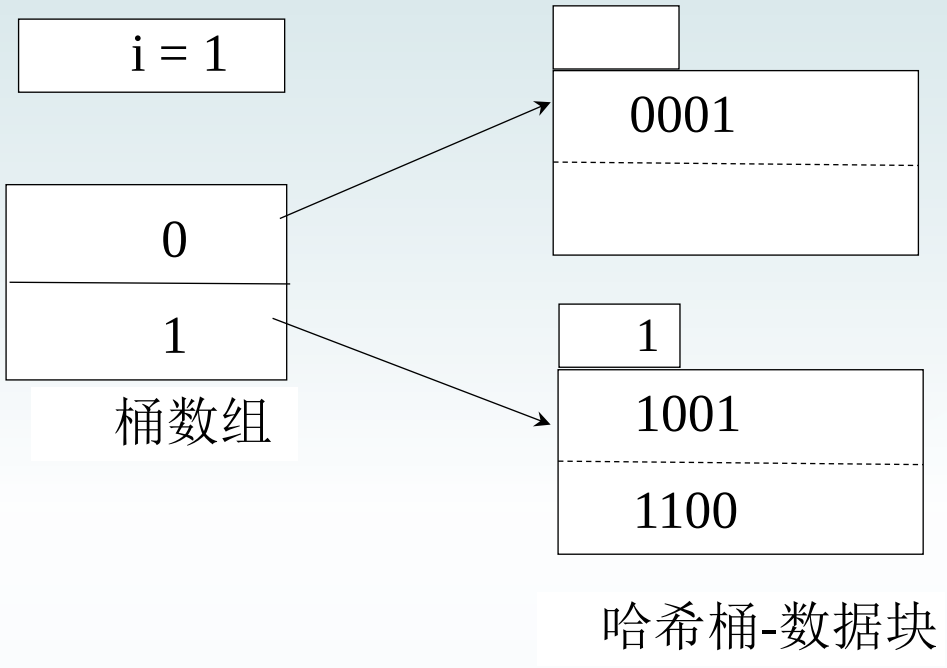
- 直接插入
- 哈希桶分裂
  - 分裂该桶的数据块，并对数据块中已有的索引项进行局部重组，并修改桶数据的指针
- 桶数组分裂
  - 桶数组的长度增加一倍

可扩展哈希索引插入操作

示例：哈希索引属性值4bits，一个桶存放至多两个数据块

- 直接插入

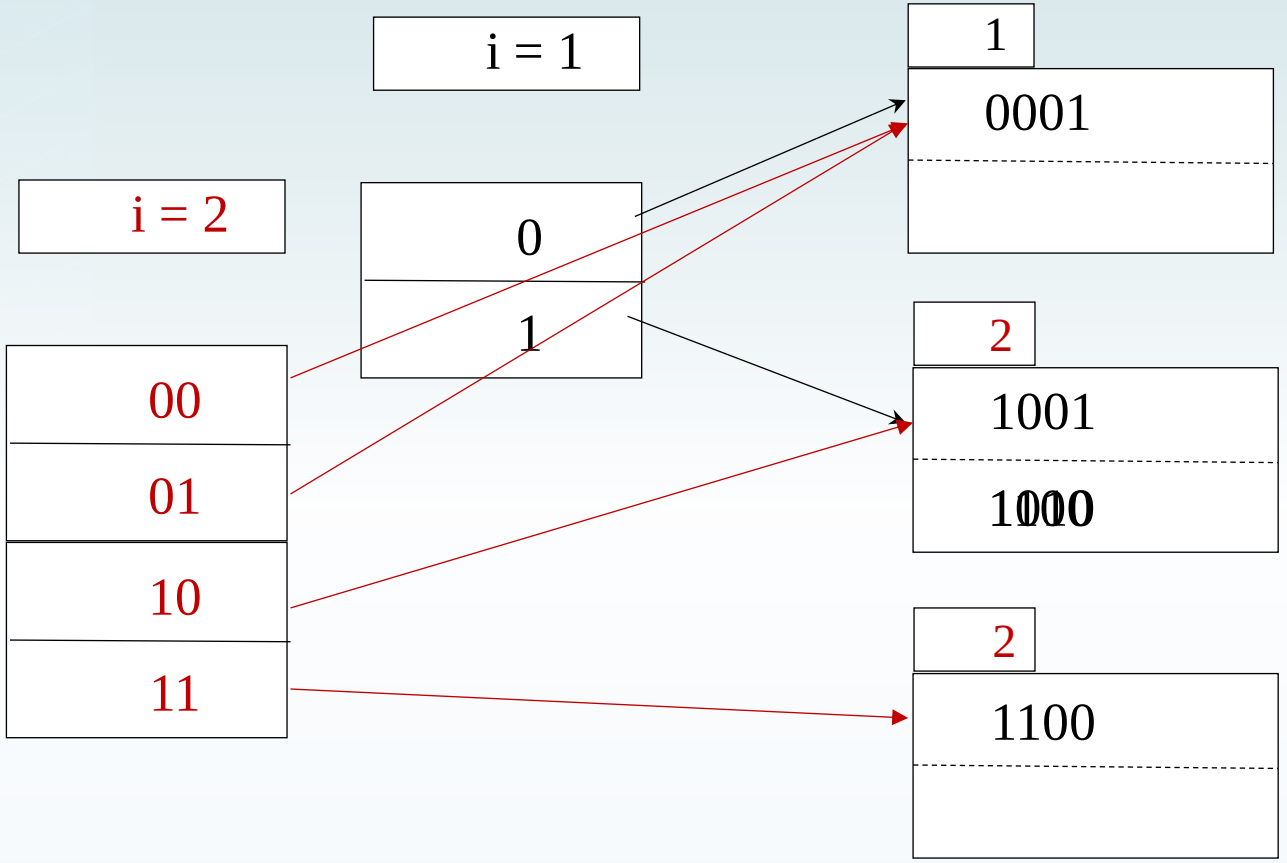
插入 1100



可扩展哈希索引插入操作

示例：哈希索引属性值4bits，一个桶存放至多两个数据块

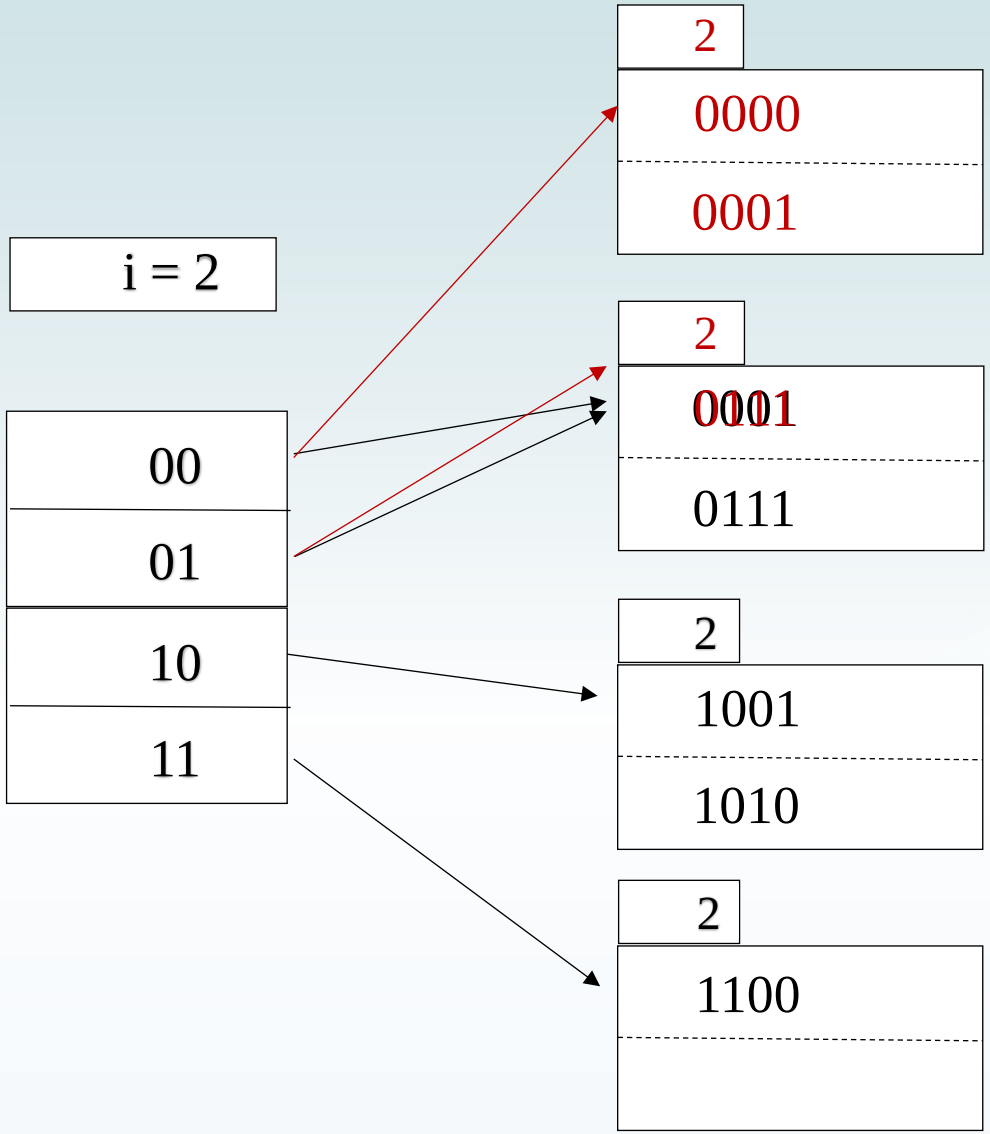
插入 1010



可扩展哈希索引插入操作

示例：

插入 0000



可扩展哈希索引删除操作

删除索可扩展哈希索引：

- 插入操作的逆操作；
- 发生索引块的合并；
- 发生桶数组的合并。

可扩展哈希索引删除操作

可扩展哈希索引：

1. 优点-在处理增量文件时：

- 浪费空间更少；
- 不需要完全重组；

2. 缺点

- 间接性，除非将目录放置内存
- 目录大小加倍增长

第6章 线性哈希索引

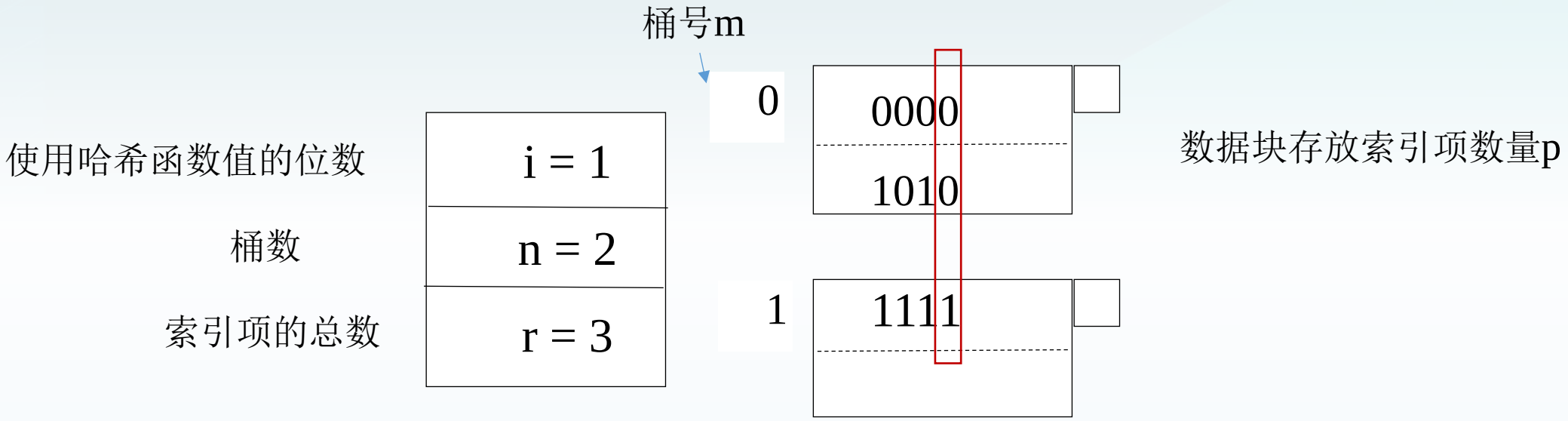
线性哈希索引

- 线性哈希索引基本结构
- 线性哈希索引插入操作

线性哈希索引

线性哈希索引基本结构：

- 线性哈希索引使用哈希函数为每个索引项键值计算出二进制序列，使用从序列右端（最后一位）开始向左的若干位来标识桶序号。



线性哈希索引插入操作

线性哈希索引插入操作：

- 直接插入
- 增加新桶，分裂旧桶
- 增加溢出块
- 提升当前被使用的散列函数的位数i

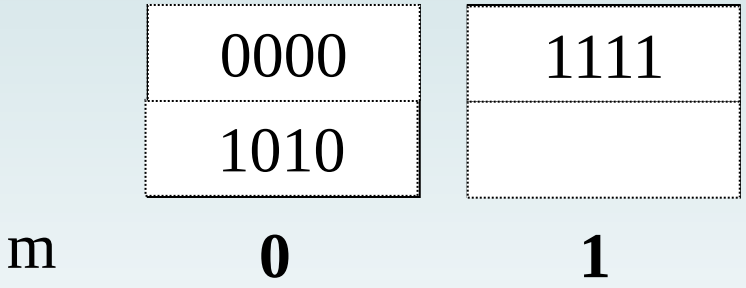
线性哈希索引插入操作

线性哈希索引插入操作：

- 直接插入
- 增加新桶，分裂旧桶
- 增加溢出块
- 提升当前被使用的散列函数的位数i

线性哈希索引插入操作

示例：插入 1010



i = 1

$\frac{r}{n*p} = \frac{3}{4} = 75\% < 85\%$ 不增加桶

max used block = 1

规则

1.使用哈希函数计算$h(K)$，取出这个二进制序列中末尾的i位表示桶号m；

2. 如果$m<n$，把索引项存放到桶m中；如果$m \geq n$，则把索引项存放到桶$m-2^{i-1}$中；

线性哈希索引插入操作

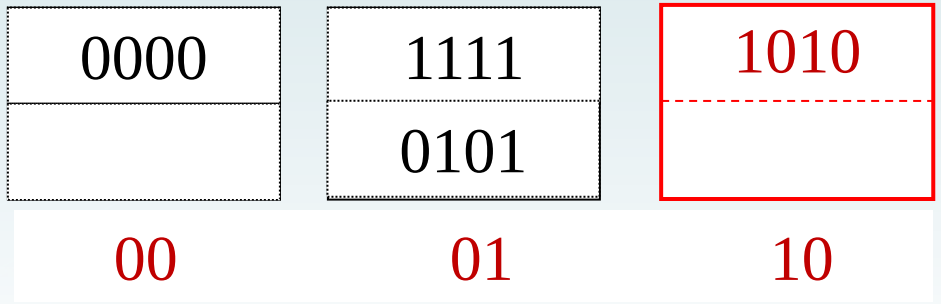
线性哈希索引插入操作：

- 直接插入
- 增加新桶，分裂旧桶
- 增加溢出块
- 提升当前被使用的散列函数的位数i

线性哈希索引插入操作

示例 插入 0101

$$\frac{r}{n*p} = \frac{4}{4} = 100\% > 85\%$$ 需要增加桶



max used block = 10

分裂**00**桶中的索引项，把键值哈希值后两位为**10**的索引项移到新桶

线性哈希索引插入操作

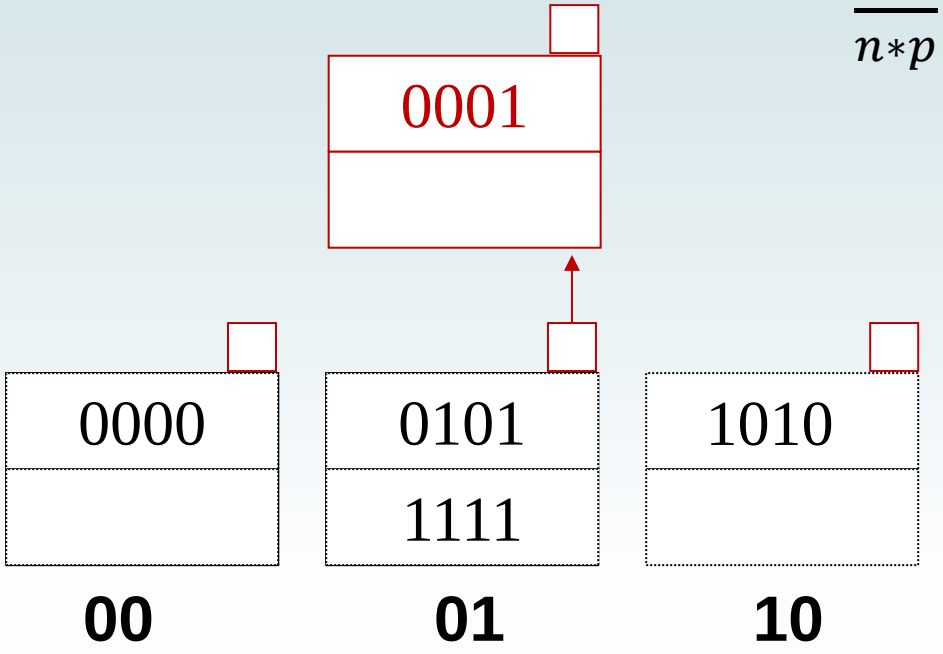
线性哈希索引插入操作：

- 直接插入
- 增加新桶，分裂旧桶
- 增加溢出块
- 提升当前被使用的散列函数的位数i

线性哈希索引插入操作

- 示例 插入 0001

$\frac{r}{n*p} = \frac{5}{6} = 83\% < 85\%$ 不需要增加桶



- 将产生溢出链

max used block = 10

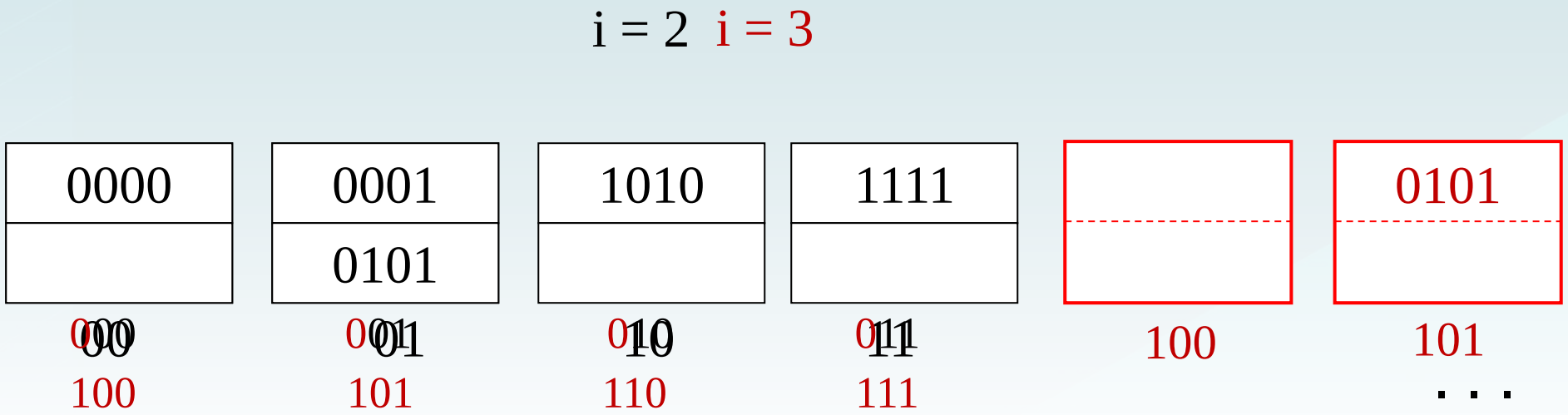
线性哈希索引插入操作

线性哈希索引插入操作：

- 直接插入
- 增加新桶，分裂旧桶
- 增加溢出块
- 提升当前被使用的散列函数的位数i

线性哈希索引插入操作

示例：提升散列函数位数



max used block = 101

可扩展哈希索引删除操作

线性哈希索引：

1. 优点-在处理增量文件时：

- 浪费空间更少；
- 没有完全重组；
- 没有像可扩展哈希那样的间接性。

2. 缺点

- 仍然存在溢出链

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

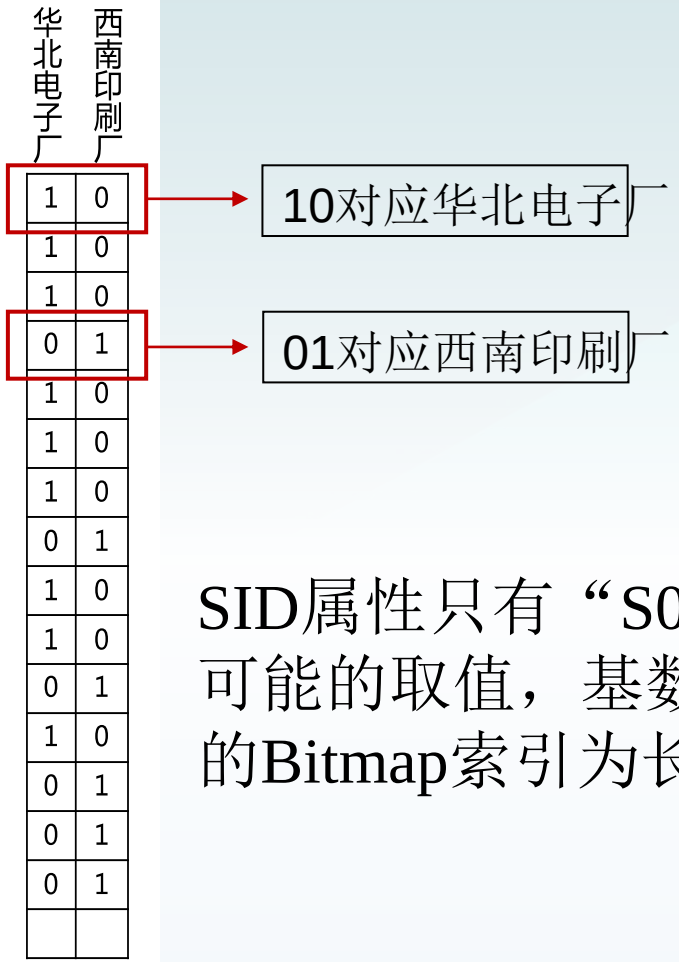
Bitmap索引

- **Bitmap索引**
  - 如果第i条记录的索引属性值为v，那么对应于值v的位向量在位置i上取值为1，其他的位向量在位置i上取值为0

Bitmap索引

- **Bitmap索引**
  - 例：在Products表的SID属性上建立Bitmap索引

PID	Pname	Price	Category	SID
P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002
P0009	人体工学鼠标	899	计算机配件	S001
P0010	移动硬盘	699	计算机配件	S001
P0011	操作系统教材	128	书籍	S002
P0012	智能手表	2999	数码产品	S001
P0013	记号笔	10	文具	S002
P0014	自动铅笔	29	文具	S002
P0015	记事本	25	文具	S002



SID属性只有“S001”和“S002”两个可能的取值，基数为2，因而SID属性上的Bitmap索引为长度为2的位向量集合。

Bitmap索引

- 例：查找Products表的所有西南印刷厂供应的商品

```
SELECT *
FROM Products
WHERE SID='S002';
```

PID	Pname	Price	Category	SID	华北电子厂	西南印刷厂
P0001	智能手机	1999	数码产品	S001	1	0
P0002	老人专用手机	899	数码产品	S001	1	0
P0003	平板电脑	1688	数码产品	S001	1	0
P0004	数据库教材	48	书籍	S002	0	1
P0005	流浪太阳	65	数码产品	S001	1	0
P0006	机械键盘	1299	计算机配件	S001	1	0
P0007	降噪耳机	999	数码产品	S001	1	0
P0008	算法导论	68	书籍	S002	0	1
P0009	人体工学鼠标	899	计算机配件	S001	1	0
P0010	移动硬盘	699	计算机配件	S001	1	0
P0011	操作系统教材	128	书籍	S002	0	1
P0012	智能手表	2999	数码产品	S001	1	0
P0013	记号笔	10	文具	S002	0	1
P0014	自动铅笔	29	文具	S002	0	1
P0015	记事本	25	文具	S002	0	1

Bitmap索引

- 例：统计华北电子厂和西南印刷厂的供应的商品数量

```
SELECT SID, Count(*)
FROM Products
GROUP By SID;
```

提示：只需要分别统计*SID*属性上Bitmap索引各个位中1的个数即可，无需访问基本表

PID	Pname	Price	Category	SID
P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002
P0009	人体工学鼠标	899	计算机配件	S001
P0010	移动硬盘	699	计算机配件	S001
P0011	操作系统教材	128	书籍	S002
P0012	智能手表	2999	数码产品	S001
P0013	记号笔	10	文具	S002
P0014	自动铅笔	29	文具	S002
P0015	记事本	25	文具	S002

华北电子厂	西南印刷厂
1	0
1	0
1	0
0	1
1	0
1	0
1	0
0	1
1	0
1	0
0	1
1	0
0	1
0	1
0	1

Bitmap索引

- 例：在Category属性也建有一个Bitmap索引

PID	Pname	Price	Category	SID
P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002
P0009	人体工学鼠标	899	计算机配件	S001
P0010	移动硬盘	699	计算机配件	S001
P0011	操作系统教材	128	书籍	S002
P0012	智能手表	2999	数码产品	S001
P0013	记号笔	10	文具	S002
P0014	自动铅笔	29	文具	S002
P0015	记事本	25	文具	S002

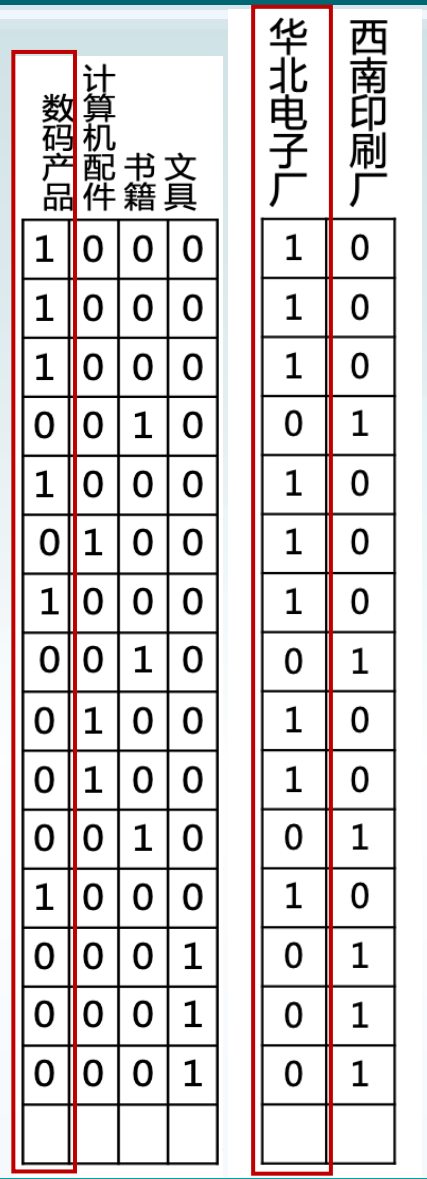
数码产品	计算机配件	书籍	文具
1	0	0	0
1	0	0	0
1	0	0	0
0	0	1	0
1	0	0	0
0	1	0	0
1	0	0	0
0	0	1	0
0	1	0	0
0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1
0	0	0	1
0	0	0	1

Bitmap索引

- 例：统计华北电子厂供应的数码产品数量

```
SELECT count(*)
FROM Products
WHERE SID='S001' AND Category='数码产品';
```

提示：只需要将SID属性Bitmap索引的第1位与Category属性Bitmap索引的第1位进行与操作，得到查询结果：(1, 1, 1,0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)



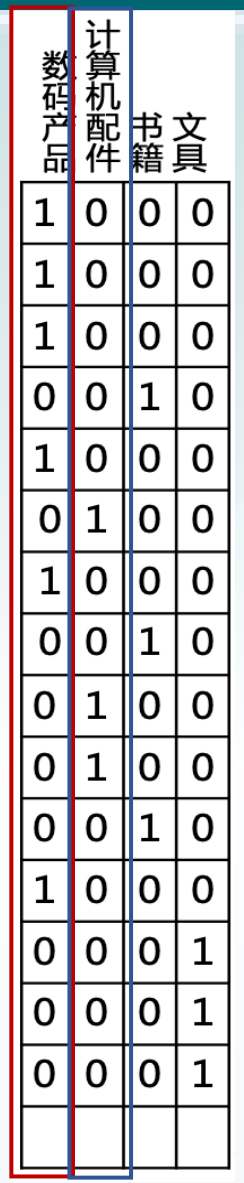
Bitmap索引

- 例：Bitmap索引处理多值查询

```
SELECT *
FROM Products
WHERE Category in (‘数码产品’, ‘计算机配件’);
```

提示：只需要对Category属性Bitmap索引的第1位和第2位进行或操作，得到结果向量：

(1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0)



数码产品	计算机配件	书籍	文具
1	0	0	0
1	0	0	0
1	0	0	0
0	0	1	0
1	0	0	0
0	1	0	0
1	0	0	0
0	0	1	0
0	1	0	0
0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1
0	0	0	1
0	0	0	1

Bitmap索引

- **Bitmap索引的优劣**
  - 优势：
    - 使用位操作来快速回答用户查询或快速定位满足条件的元组集合，减少对基本表的全表扫描，提高查询效率
  - 劣势：
    - Bitmap索引的大小与列的基数成正比，基数大的列其Bitmap索引会非常庞大，因此它只适用于基数小的属性列

Bitmap索引

- **编码Bitmap索引（Encoded Bitmap Index）**

改进标准Bitmap索引：通过对属性值编码，减少索引位向量的个数，从而能够应对有较高基数的列

Bitmap索引

- **编码Bitmap索引（Encoded Bitmap Index）**

例：Products表Category属性的编码Bitmap索引

PID	Pname	Price	Category	SID
P0001	智能手机	1999	数码产品	S001
P0002	老人专用手机	899	数码产品	S001
P0003	平板电脑	1688	数码产品	S001
P0004	数据库教材	48	书籍	S002
P0005	流浪太阳	65	数码产品	S001
P0006	机械键盘	1299	计算机配件	S001
P0007	降噪耳机	999	数码产品	S001
P0008	算法导论	68	书籍	S002
P0009	人体工学鼠标	899	计算机配件	S001
P0010	移动硬盘	699	计算机配件	S001
P0011	操作系统教材	128	书籍	S002
P0012	智能手表	2999	数码产品	S001
P0013	记号笔	10	文具	S002
P0014	自动铅笔	29	文具	S002
P0015	记事本	25	文具	S002

0	0
0	0
0	0
1	0
0	0
0	1
0	0
1	0
0	1
0	1
1	0
0	0
1	1
1	1
1	1

编码映射表

数码产品	00
计算机配件	01
书籍	10
文具	11

Bitmap索引

- **编码Bitmap索引的优劣**
  - 优势：减少索引位个数
    - 如果索引属性列的基数为K，标准Bitmap索引所需要的位向量个数为K个，而编码Bitmap索引所需要的位向量个数仅为$\log_2 K$个
  - 劣势：查询时需要访问所有位向量才能完成
    - 例：在学生表中查找信息管理与信息系统专业和信息安全专业的所有学生，需要扫描所有位向量，从中查找编码为10或00的元组

第6章 索引

- 6.1 顺序表的索引
- 6.2 辅助索引
- 6.3 B+树索引
- 6.4 哈希索引
- 6.5 Bitmap索引
- 6.6 小结

小结

- 顺序表索引建立在按索引属性值顺序存放的关系表上。顺序表索引分为稠密索引和稀疏索引两类。辅助索引是建立在关系表的非排序属性上的索引。辅助索引是一种稠密索引。
- B+树就是为了解决大型索引的组织和维护而产生的一种索引结构，具有查找效率高、按不同值查找的性能平衡、易于维护等特点。
- 哈希索引是另一类能够实现快速查找的索引结构。哈希函数和哈希表是哈希索引的两个关键要素。
- Bitmap索引由位向量集合组成，利用位操作快速完成查询请求或快速定位满足条件的元组集合。编码Bitmap索引通过对属性值编码减少索引位向量的个数，以应对有较高基数的列。