数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第4章作业问题

（Chap4-7）请用SQL的GRANT 和REVOKE语句(加上视图机制)完成以下授权定义或存取控制功能:

(3)每个职工只对自己的记录有SELECT 权力。

```
GRANT SELECT ON 职工
WHEN USER()=NAME
TO ALL;
```

//网上的答案，语法是kingbase的，

//大多DBMS都不支持

如果是mysql，

```
CREATE VIEW V1 AS
  SELECT * FROM TABLE 职工 WHERE 姓名=USER()
;
GRANT SELECT ON V1 TO PUBLIC;
```

第4章作业问题

（7）用户杨兰具有从每个部门职工中SELECT 最高工资、最低工资、平均工资的权力，他不能查看每个人的工资。

```
CREATE VIEW 部门工资 AS
    SELECT 部门.名称,MAX(工资),MIN(工资),AVG(工资)
    FROM 职工,部门
    WHERE 职工.部门号=部门.部门号
    GROUP BY 职工.部门号
GRANT SELECT ON 部门工资 TO 杨兰;
```

//有同学在GRANT语句中建视图，这是各个DBMS都不支持的。

第八章 关系数据库引擎基础

8.1 数据库存储

8.2 缓冲池

8.3 索引

8.3.1 B 树概述

8.3.2 B+树结构

8.3.3 聚簇索引

8.3.1 B 树概述

- B树：一种特定的数据结构，具有平衡的多分树结构。

- “B树家族”，可用于泛指一类平衡树的数据结构。
  - B-Tree (1971)
  - B+Tree (1973)
  - B* Tree (1977)
  - Blink -Tree (1981)

- B树是专门为外部存储器设计的，如磁盘，它对于读取和写入大块数据有良好的性能，所以一般被用在文件系统及数据库中。
- 关系数据库常见索引结构：B+树

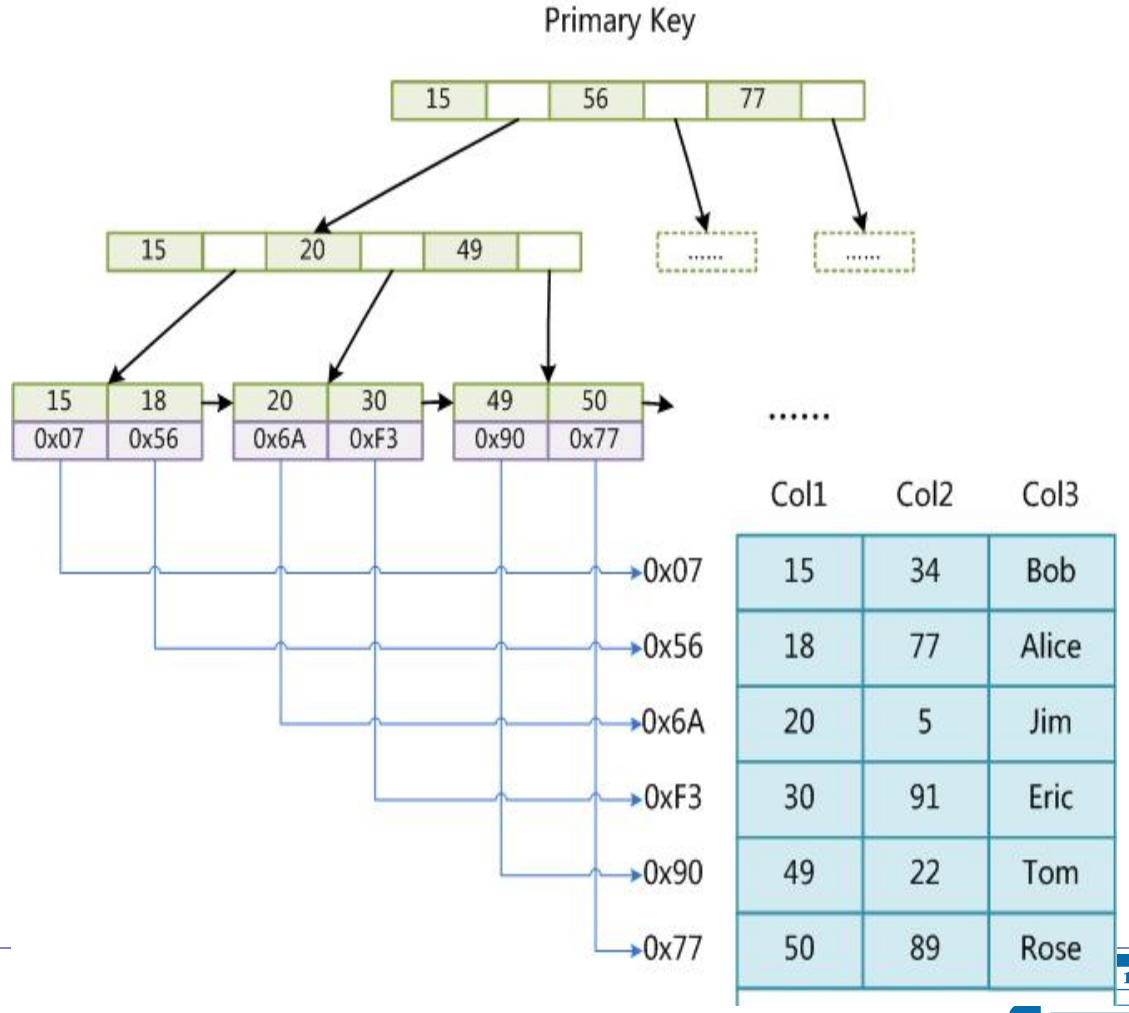
B+树性质

- B+树是一种自平衡的树型数据结构：

  它保持数据排列，允许在O(log n)的复杂度内进行搜索、顺序访问、插入和删除；

  - 二叉搜索树的泛化，一个结点可以有2个以上的子结点；
  - 针对读写大数据块的系统进行优化。

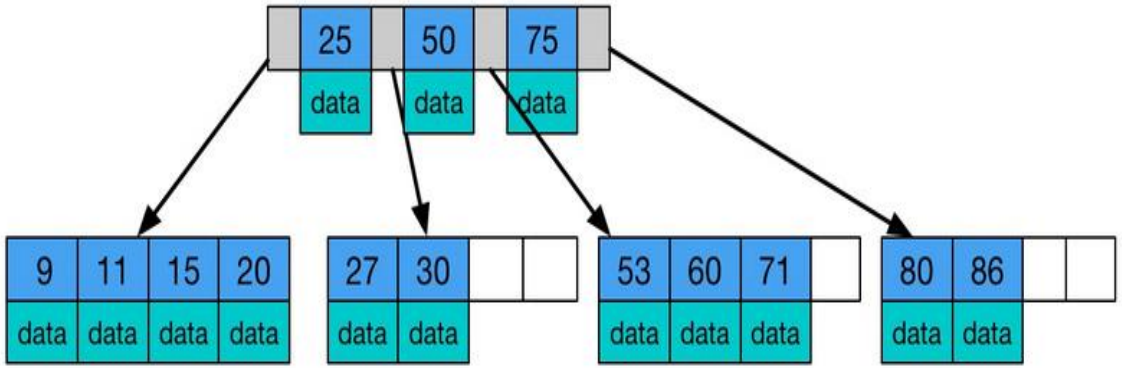
空间局部性原理：如果一个存储器的某个位置被访问，那么将它附近的位置也会被访问。



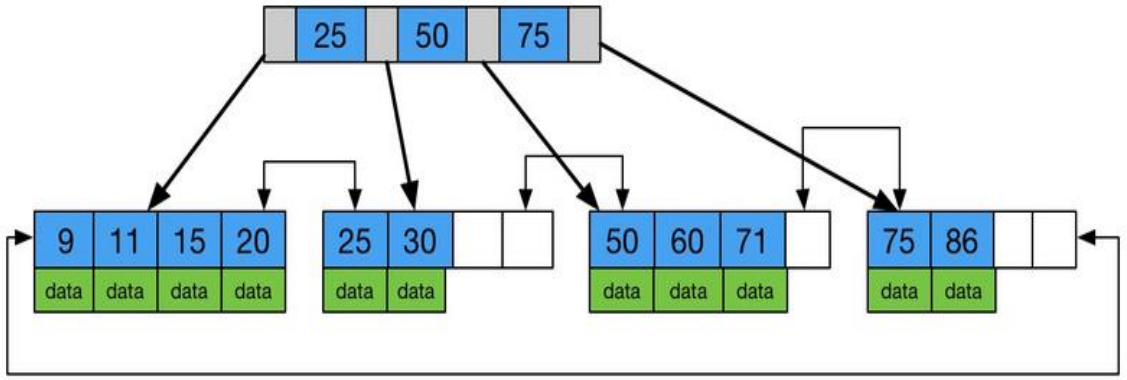
B-树 vs. B+树

1972的原始B-树在所有结点中存储“键+值”，每个键只在树中出现一次，存储空间更高效。

B+树只在叶子结点中存储值，内部结点无 data 域，仅用于引导搜索过，内节点每个节点能索引的范围更大更精确程。B+树更适合外部存储。



简化 B- 树

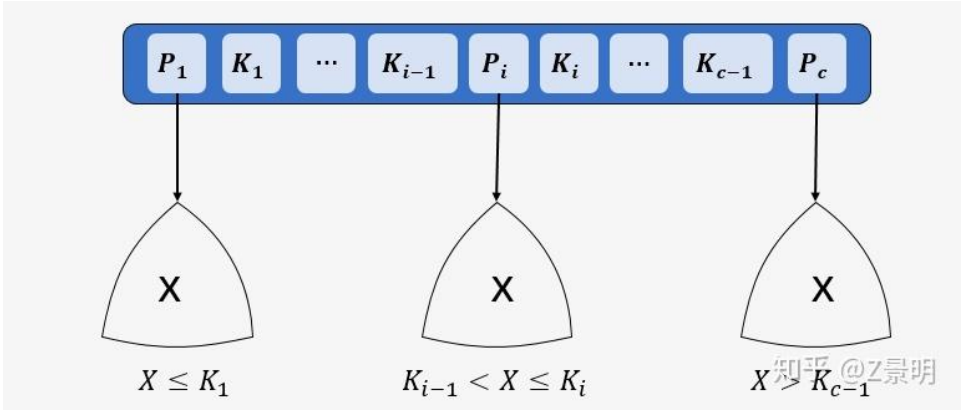


简化 B+ 树

8.3.2 B+树结构

阶为M的**B+树内部结点的结构**：

1. 对于每一个形如： <P1, K1, P2, K2, ……., Pc-1, Kc-1, Pc>的内部结点，
   其中 c<M ， 每一个Pi 表示指向子树根结点的指针， Ki 表示关键字值；
2. 内部结点的关键字由小到大有序排列
3. 对于一个位于Pi 所指向的子树中的结点X 而言，
   当 1<i<c 时， 均有 Ki-1<X<=Ki .
   当 i=c 时， X>Kc-1 .
   当 i=1 时， X<=K1 .



4. 每一个内部结点最多有M 个指向子树的指针， 即 c 最大取 M .
5. 根结点至少包含两个指向子树的结点指针， 即对于根结点而言 2<=c<=M ；
   除了根之外的每个结点都包含最少 ceil(M/2) 个指向子树的指针， 完全平衡树。
6. 如果任意一个内部结点包含 c 个指向孩子结点的指针且 c<=M ， 则该结点包含 c-1 的关键字。

8.3.2 B+树结构

□ 阶为 N 的 **B+树叶子结点的结构**：

1. 对于每一个形如：<<K1,D1>, <K2,D2>,……, <Kc-1,Dc-1>, Pnext>的叶子结点，其中 c<=N， Di 是一个数据指针（指向磁盘上的值等于 Ki 的真实记录的指针，或者包含记录 Ki 的磁盘文件块），Ki 是一个关键字，Pnext 表示 B+树中指向下一个叶子结点的指针。

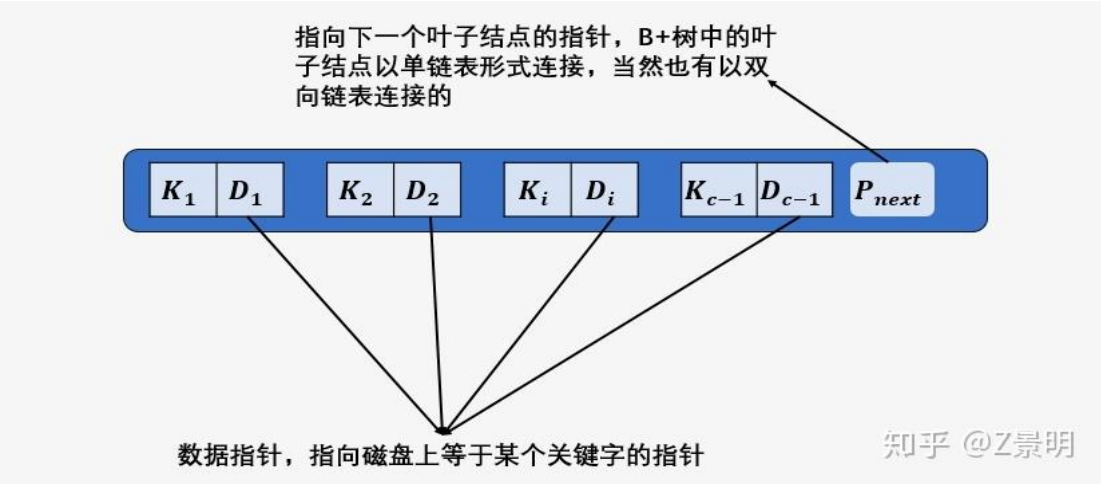
2. 对任意一个叶子结点均有：

   K1 < K2 < … < Kc-1,

   c<=N .

3. 每一个叶子结点至少包含 ceil(M/2) 个值.

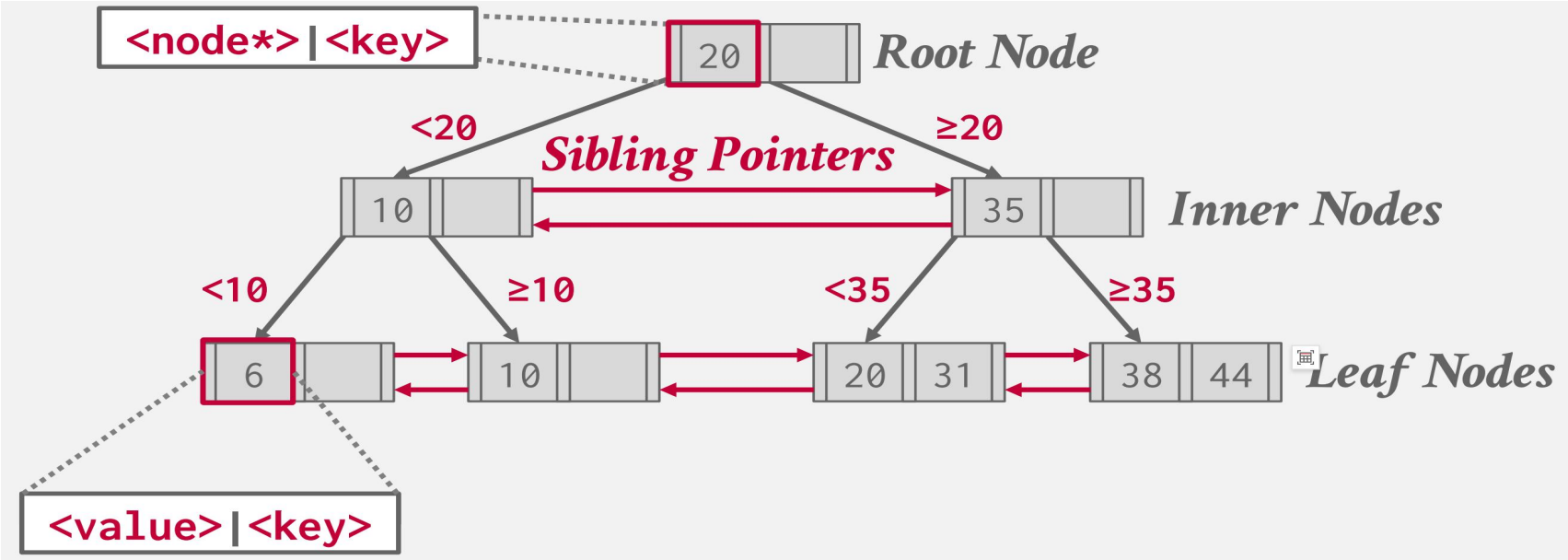
4. 所有的叶子结点在同一层。



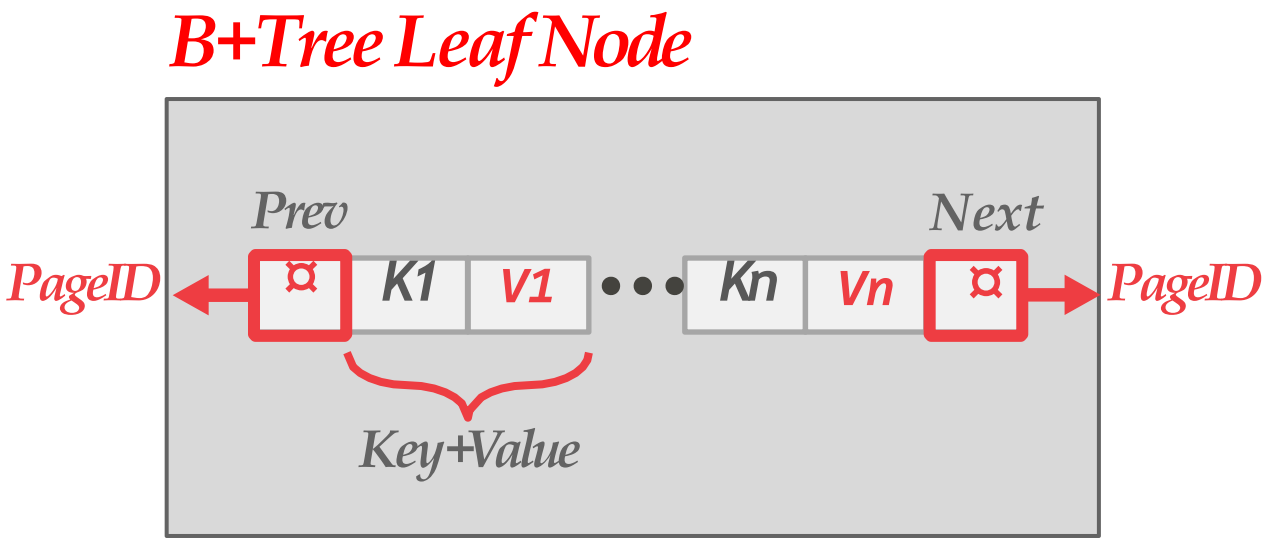
使用 Pnext 指针可以遍历所有的叶子结点，就和单链表一样，从而实现对磁盘上记录的有序访问。

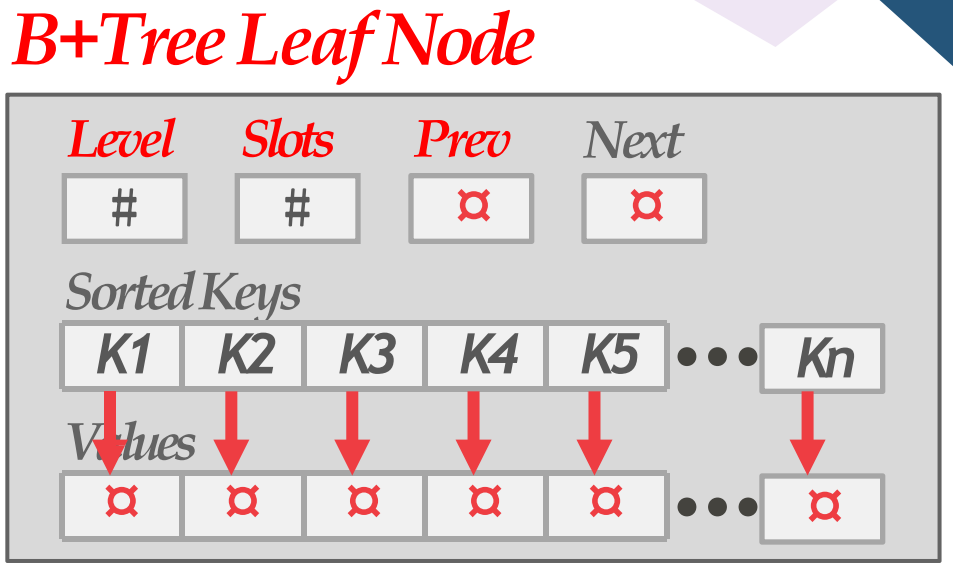
8.3.2 B+树结构

- 所有关键字K都出现在 叶子结点 的链表中，即数据只能在叶子节点（也称为：稠密索引），且链表中的关键字(数据)恰好是有序的；
- 非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储（关键字）数据的数据层。



B+树示例——叶子节点





B+树叶子结点的值：

方法一：记录ID

指向索引项所对应元组位置的指针。

方法二：元组数据

元组的实际数据存储在叶子结点中；

二级索引则必须采用将记录ID作为叶节点值的方式。

选择条件（Selection Conditions）

- 如果查询提供了B+树搜索关键字的任何属性的值，DBMS就可以使用B+树索引。
  - 全值匹配
  - 匹配最左前缀
  - 匹配列前缀
  - 匹配范围值
  - 只访问索引的查询

【例】<a, b, c>上的索引，可以支持的查询条件

(a=5 and b=3)；

(b=3)

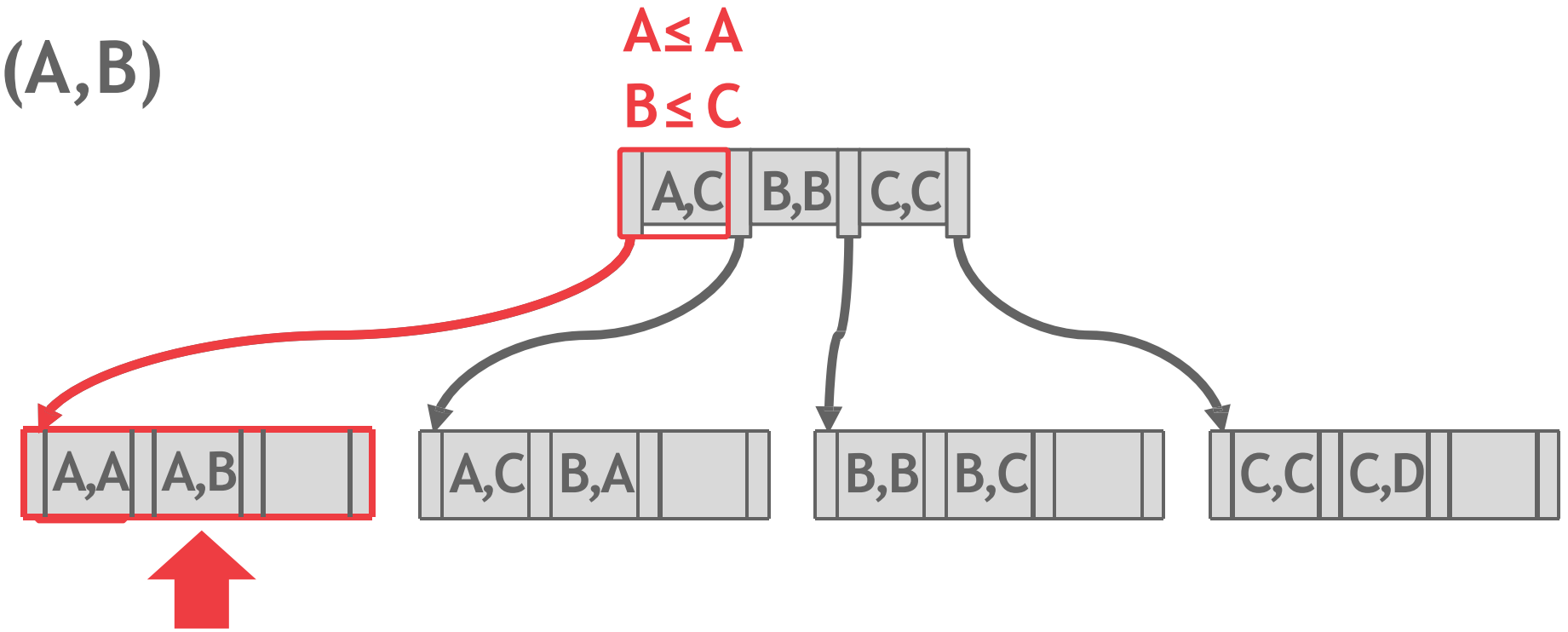
并非所有DBMS都支持如此。

PS：如果采用哈希索引，则需要搜索关键字中的所有属性。

选择条件（Selection Conditions）

- 索引的分类：主键索引 / 唯一索引 / 普通索引 / 全文索引 / 组合索引

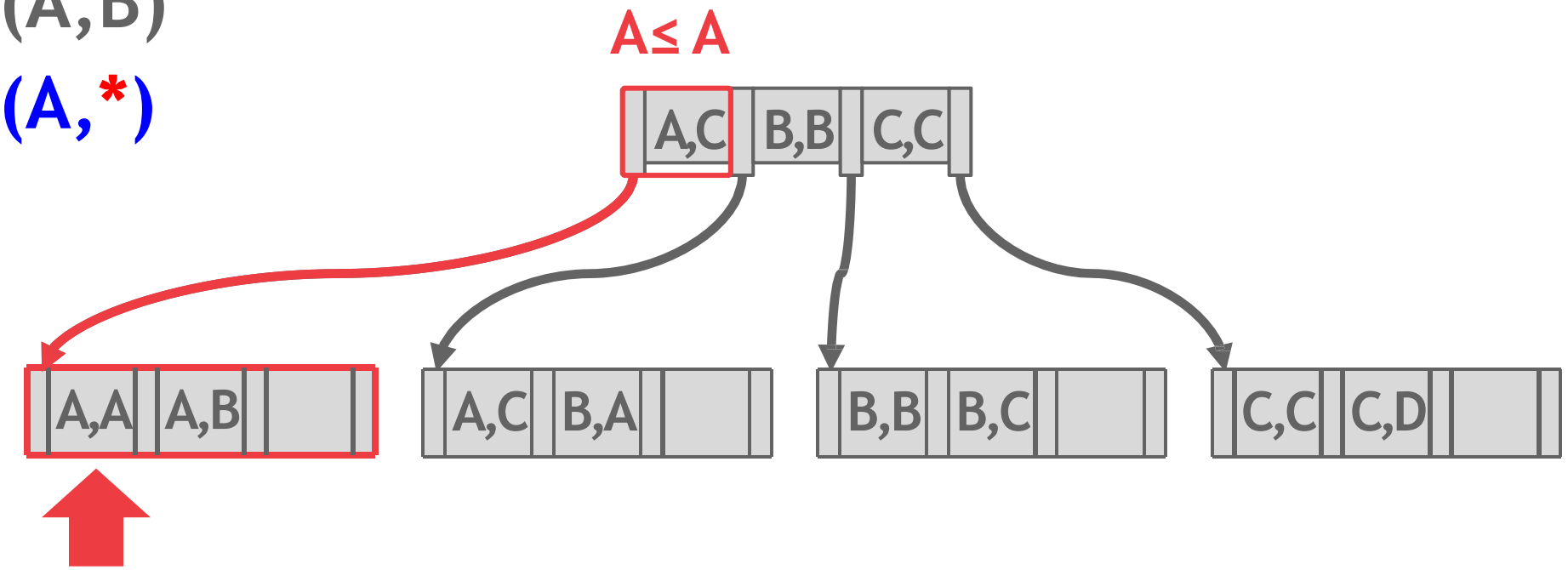
Find Key=(A,B)



选择条件（Selection Conditions）

Find Key=(A,B)

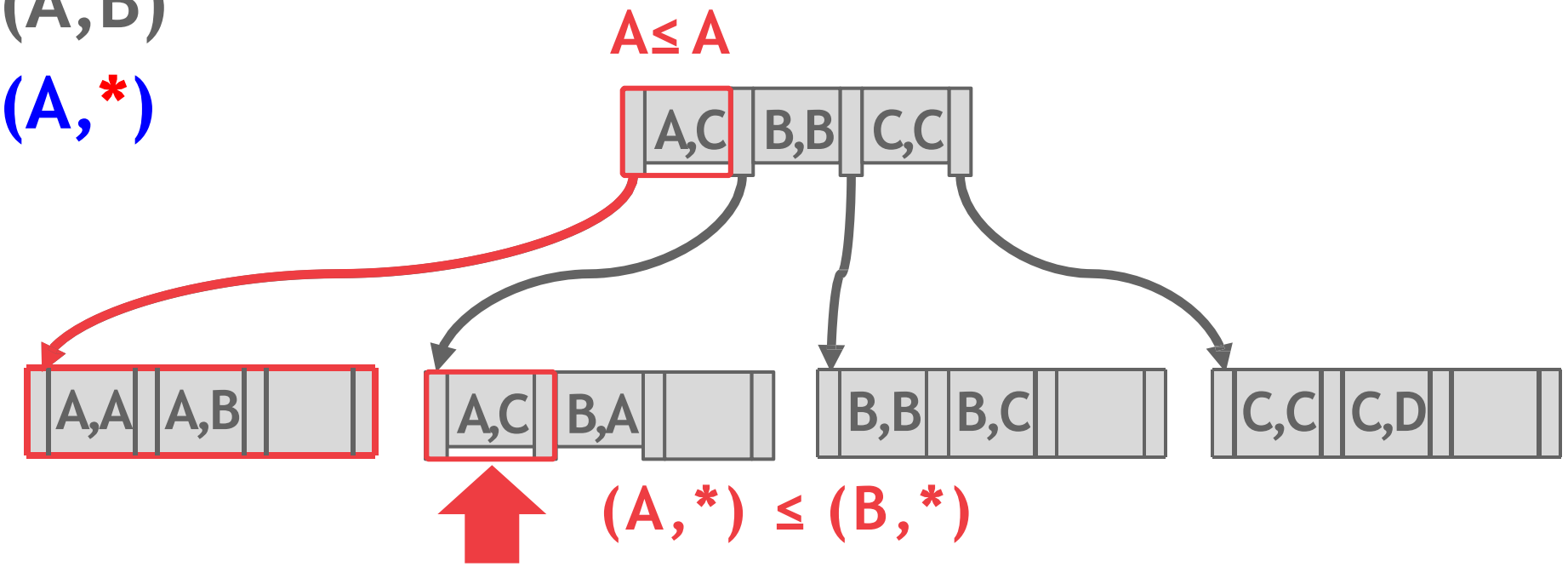
Find Key=(A,*)



选择条件（Selection Conditions）

Find Key=(A,B)

Find Key=(A,*)

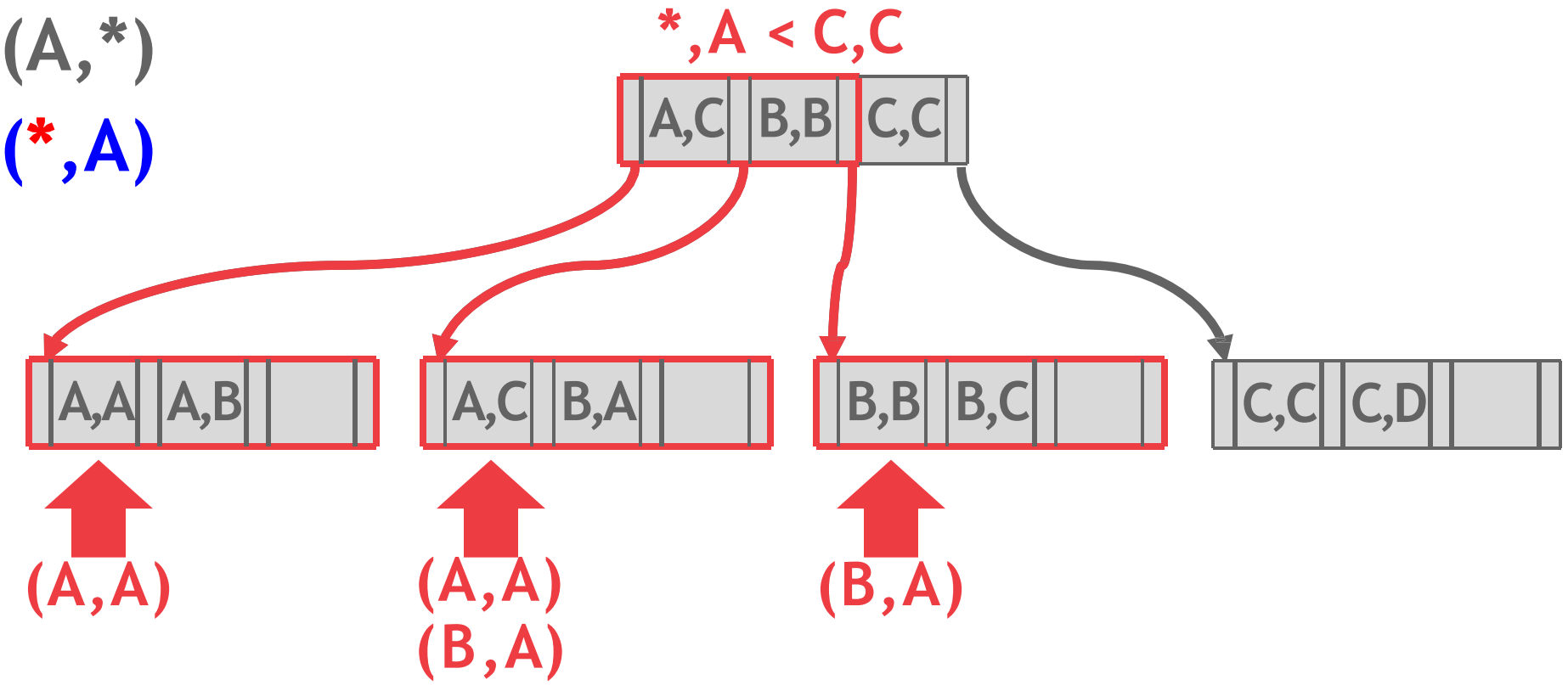


选择条件（Selection Conditions）

Find Key=(A,B)

Find Key=(A,*)

Find Key=(*,A)



B+树-插入操作

1）若为空树，创建一个叶子结点，将记录插入其中，此叶子结点为根结点，插入操作结束。

2）针对叶子结点：

（2.1）根据key值确定叶子结点L，将排序后的关键字插入此结点L；

（2.2）如果空间充足（即结点L含有的关键字数目小于阶数m），则插入结束；

（2.2）否则将结点L分裂为左右两个叶子结点（L和L2）：

左叶子结点包含前m/2个记录；

右叶子结点包含剩下的记录；

将第m/2+1个记录的key进位到父结点中，并完成在父（内）节点的插入（可能引起分裂递归）。

B+树操作的可视化呈现：

http://cmudb.io/btree

由旧金山大学的助理教授David Galles开发

B+树-删除操作

如果叶子结点中没有相应的key，则删除失败。否则执行：

（1）从根结点出发找到该关键字所在结点L，删除该关键字；

（2.1）如果结点L的关键字数目不少于M/2，则删除完成，结束；

（2.2）如果结点L仅有M/2-1个关键字数目，向兄弟结点借一个关键字（兄弟结点指与L有相同父结点的相邻结点）；

（2.3）否则将结点L与其兄弟结点合并，合并时需要删除父结点中的关键字（指向L或其兄弟结点，可能引起删除内节点的级联合并）。

B+树-重复键（Duplicate Keys）

在B+树中有两种允许重复键的方法：

方法一：附加记录ID作为键的一部分

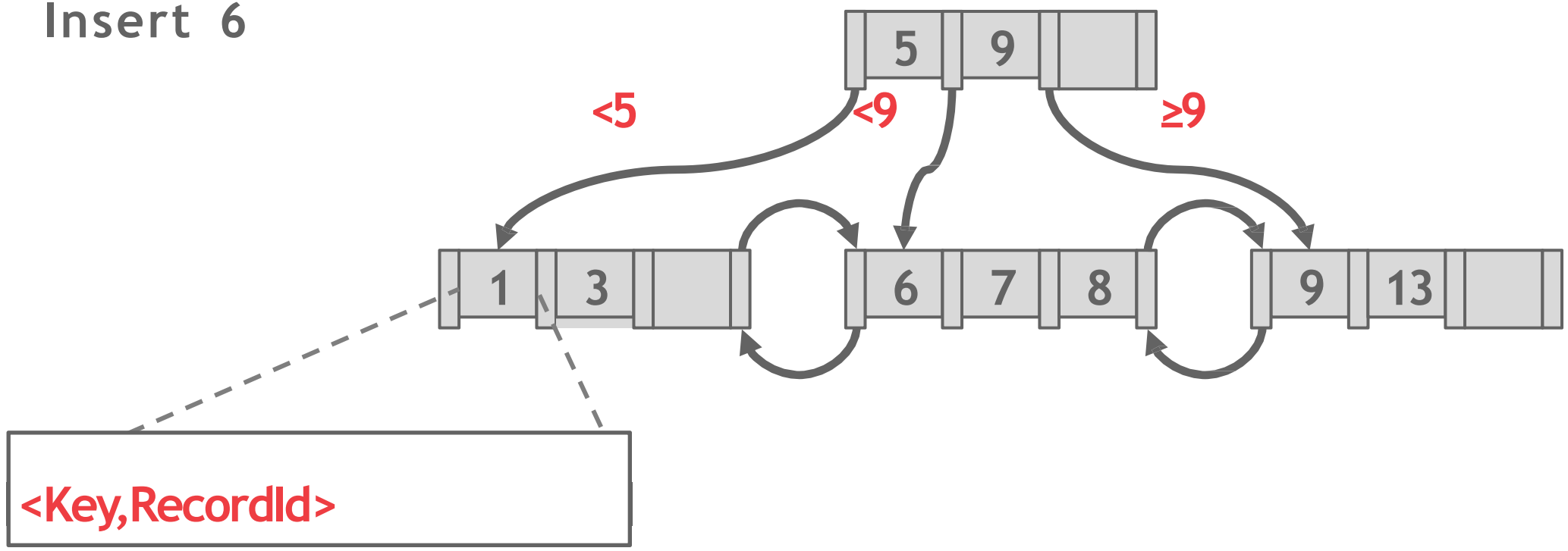
由于每个元组的记录ID是唯一的，因此确保了所有键都是可识别的。

方法二：允许叶结点溢出至包含重复键的溢出结点

该方法的维护和修改较为复杂。

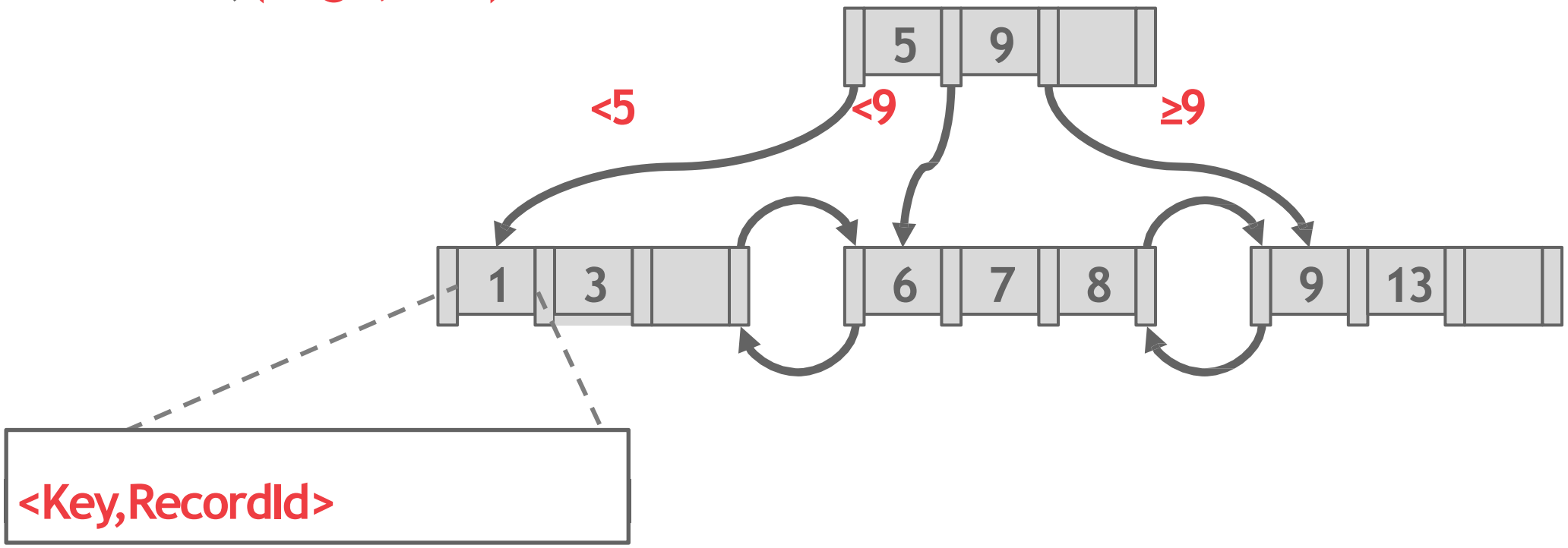
B+树-附加记录ID

Insert 6



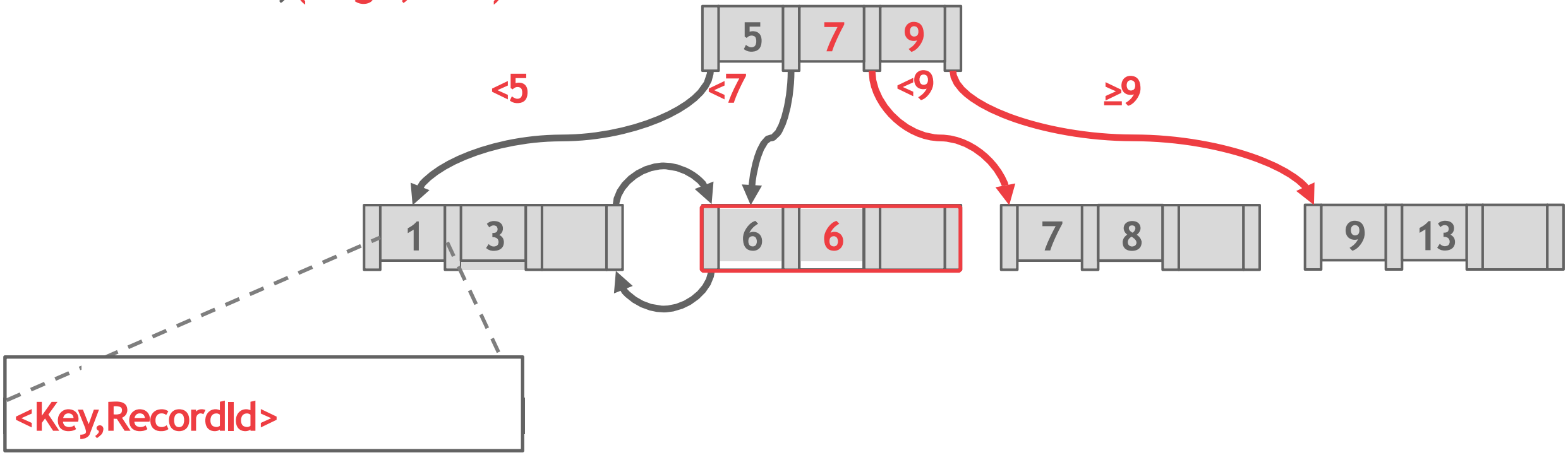
B+树-附加记录ID

Insert <6,(Page,Slot)>



B+树-附加记录ID

Insert <6,(Page,Slot)>

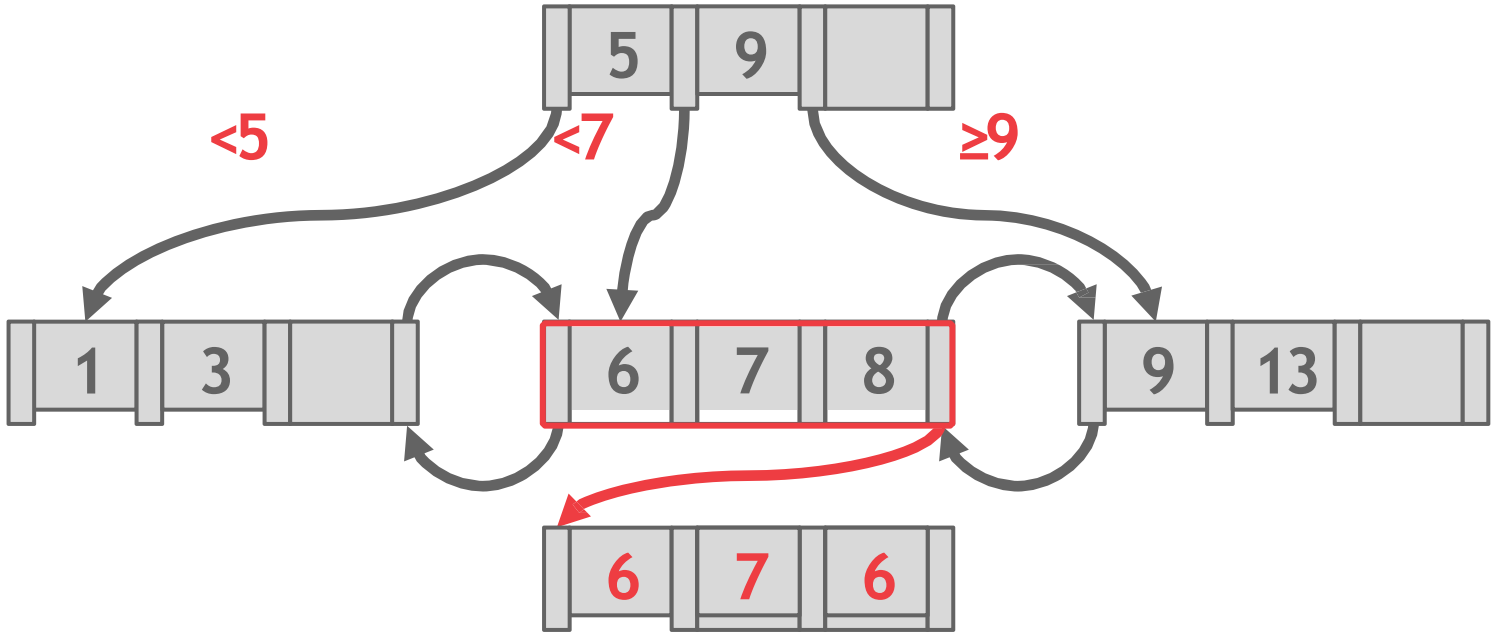


B+树-叶结点溢出

insert 6

insert 7

insert 6



索引适用场景

- 选择索引存取方法的一般规则：
  - 如果一个(或一组)属性经常在查询条件中出现，则考虑在这个(或这组)属性上建立索引(或组合索引)
  - 如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引
  - 如果一个(或一组)属性经常在连接操作的连接条件中出现，则考虑在这个(或这组)属性上建立索引
- 关系上定义的索引数过多会带来较多的额外开销：
  - 维护索引的开销
  - 查找索引的开销

8.3.3 聚簇索引（Clustered Indexes）

- 关系按照主键的排列顺序存储

  可以是基于堆的存储，或者索引组织的存储。

- 聚簇索引：存储的数据在物理内存上也按照主键进行排序。

- 一些DBMS使用聚簇索引

  如果一个关系没有主键，DBMS则会自动生成一个隐藏的行ID主键。

- 有些DBMS两种方式都使用。

聚簇索引和非聚簇索引

- 聚簇索引物理有序，非聚簇索引逻辑有序，物理无序。

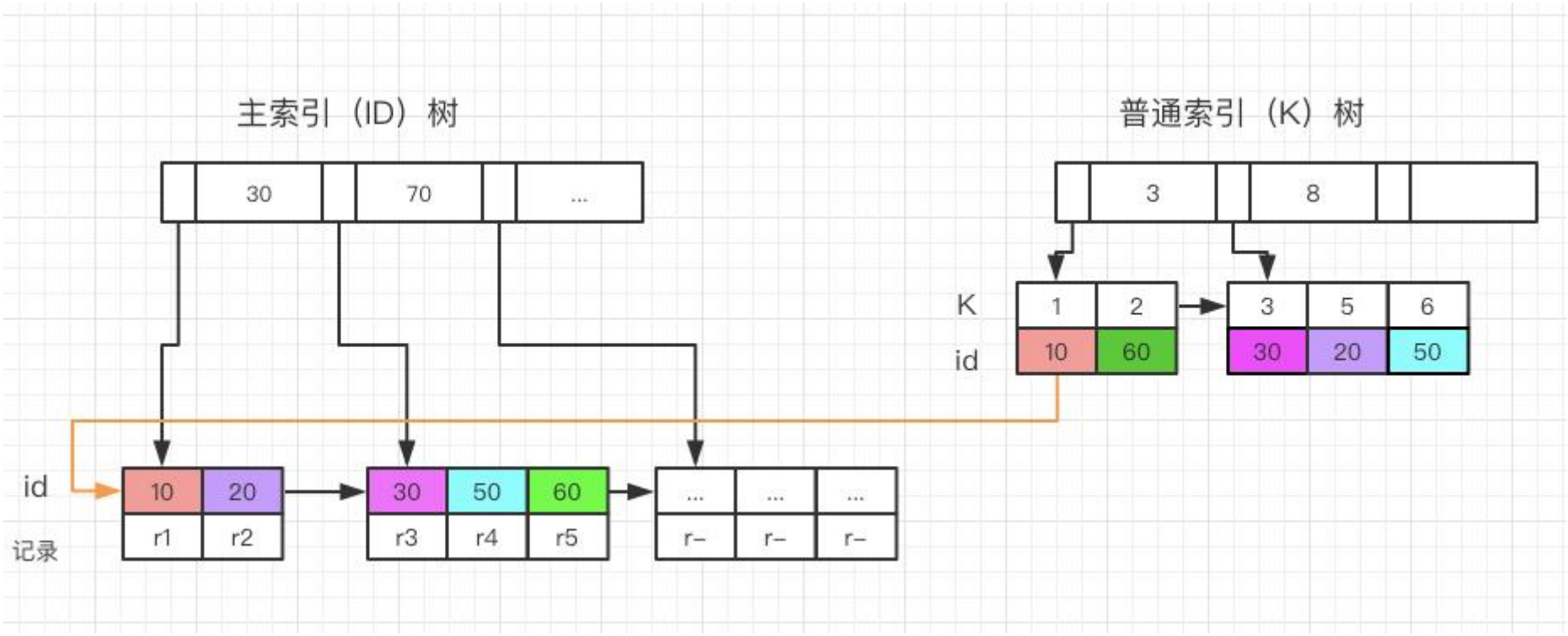
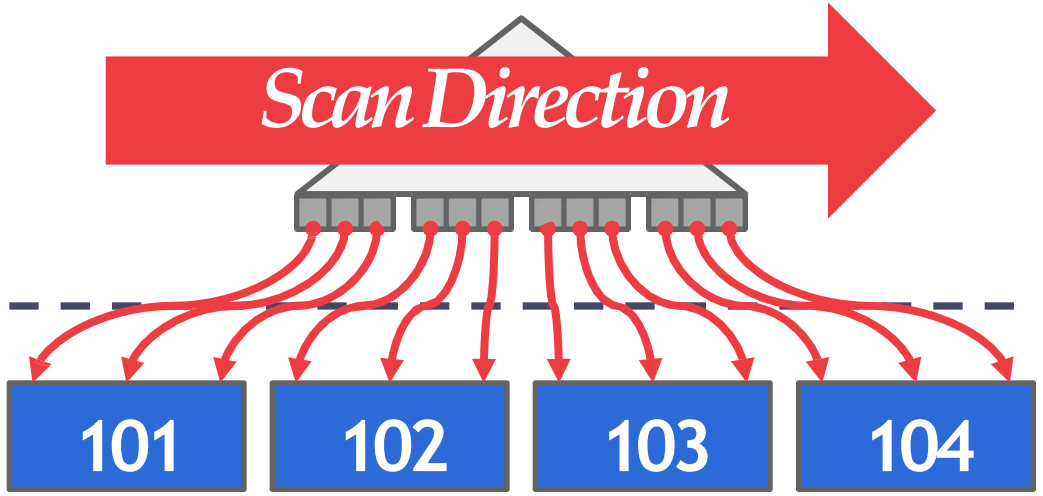


图 Mysql聚簇索引与非聚簇索引的关系示意图

堆聚簇（Heap Clustering）

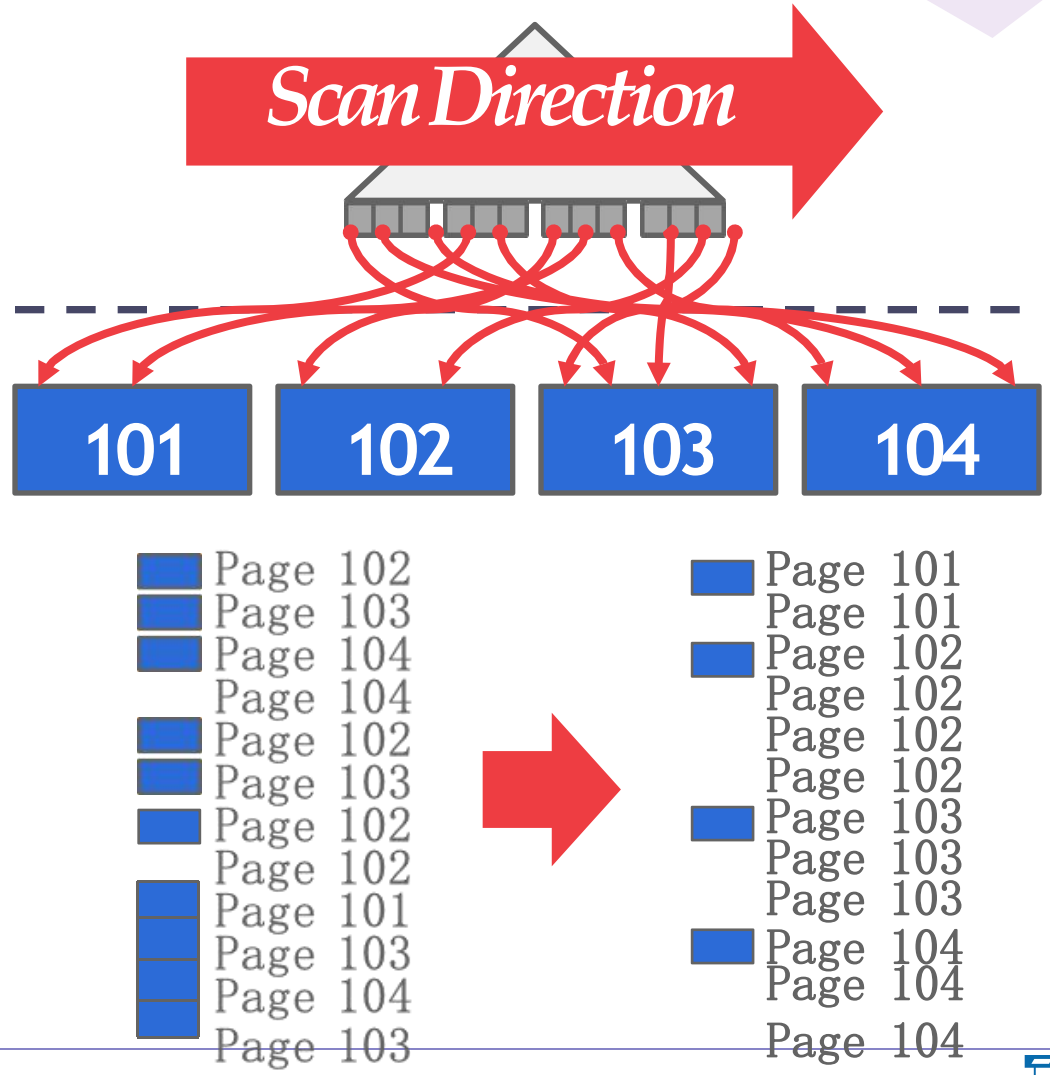
元组在heap页面集合中按照聚簇索引指定的顺序排序。

如果使用聚集索引的属性来访问元组，那么DBMS可以直接跳转至目标页。



索引扫描页面排序（Index Scan Page Sorting）

- 在非聚集索引中按照元组出现的顺序检索元组是十分低效的。
- DBMS可以先找出所需要的所有元组，然后根据其页ID对元组进行排序。



聚簇适用场景

- 设计候选聚簇：
  - 对经常在一起进行连接操作的关系可以建立聚簇；
  - 如果一个关系的一组属性经常出现在相等比较条件中，则该单个关系可建立聚簇；
  - 如果一个关系的一个(或一组)属性上的值重复率很高，则此单个关系可建立聚簇。即对应每个聚簇码值的平均元组数不太少。太少则聚簇的效果不明显。
- 优化聚簇设计：
  - 从聚簇中删除经常进行全表扫描的关系；
  - 从聚簇中删除更新操作远多于连接操作的关系；
  - 不同的聚簇中可能包含相同的关系，一个关系可以在某一个聚簇中，但不能同时加入多个聚簇；
  - 从这多个聚簇方案(包括不建立聚簇)中选择一个较优的，即在这个聚簇上运行各种事务的总代价最小。

聚簇适用场景

- 聚簇的适用范围：

1. 既适用于单个关系独立聚簇，也适用于多个关系组合聚簇

【例】假设用户经常要按系别查询学生成绩单，这一查询涉及学生关系和选修关系的连接操作，即需要按学号连接这两个关系：

为提高连接操作的效率，可以把具有相同学号值的学生元组和选修元组在物理上聚簇在一起。相当于把多个关系按“预连接”的形式存放，从而大大提高连接操作的效率。

2. 当通过聚簇码进行访问或连接是该关系的主要应用，与聚簇码无关的其他访问很少或者是次要的时，可以使用聚簇。

尤其当SQL语句中包含有与聚簇码有关的ORDER BY，GROUP BY，UNION，DISTINCT等子句或短语时，使用聚簇特别有利，可以省去对结果集的排序操作。

聚簇

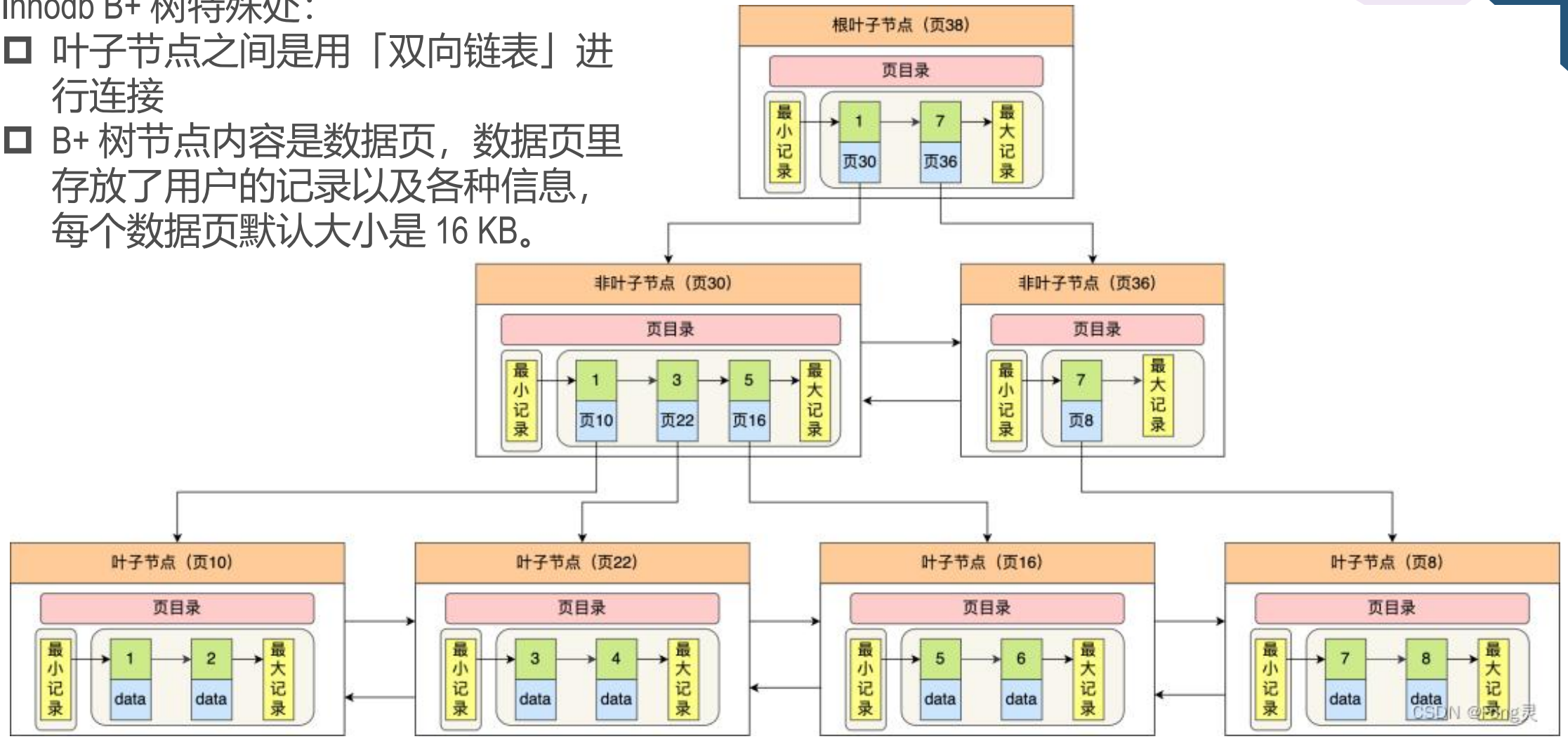
- 聚簇的局限性：

1. 聚簇只能提高某些特定应用的性能。
2. 建立与维护聚簇的开销相当大。
   - 对已有关系建立聚簇，将导致关系中元组移动其物理存储位置，并使此关系上原有的索引无效，必须重建。
   - 当一个元组的聚簇码改变时，该元组的存储位置也要做相应移动。

Mysql B+树

Innodb B+ 树特殊处：

- 叶子节点之间是用「双向链表」进行连接
- B+ 树节点内容是数据页，数据页里存放了用户的记录以及各种信息，每个数据页默认大小是 16 KB。



本章小结

- 掌握面向磁盘的DBMS数据库存储结构：
  - 文件级：堆文件组织（链表、页目录）；
  - 页级：页ID、页头、数据区（面向元组型、日志结构型）
  - 元组：槽页、元组组织（定长/变长元组存储方式）
- NSM和DSM存储模型的差异。
- 掌握缓冲池工作原理
  - 缓冲池结构（页表的组成）
  - 缓冲池替换策略（LRU和LRU-K）
- 了解B-树与B+树的区别；掌握B+索引结构，插入、删除原理

本章作业

1. 简述数据库堆文件的链表和页目录两种组织方式中，空闲空间分别是如何组织的?

2. 简述面向元组型的页设计中槽页方案的基本思想，并简要说明槽页方案相对于数组式的元组存储方案的好处。

3. 对于如下B+树，阶数为4，插入7，请问该B+树将如何变化？删除14，该B+树又如何变化?