

第5章 数据库的存储结构

2012. 03



目录 Contents

■ 5.1 数据库存储结构

- 多级存储
- 物理结构
- 逻辑结构

■ 5.2 关系数据库中表的典型存储机制

- 索引
- 散列
- 簇集



5.1 数据库存储结构

- 一、多级存储
- 目前，用内存作为数据库的存储介质是不合适的。
 - 内存的容量尚不够存储全部的数据；
 - 内存一般属易失存储器(Volatile Storage)，不适合用来存储持久数据
 - 内存存储单位数据成本要比外存高得多
- 因此，主要采用多级存储器。
 - 二级存储：内存—外存
 - 三级存储



5.1 数据库存储结构

■ 三级存储结构

- 第一级：主存储器 (main memory)
 - 高速缓冲存储器 (cache)
 - 主存储器 (memory)
- 第二级：磁盘存储器 (secondary storage)
 - 也称为：二级存储器 或 次级存储器。
- 第三级：辅助存储器 (tertiary storage)
 - 磁带存储器
 - 自动光盘机
 - 是一种辅助存储设备，也称三级存储器。

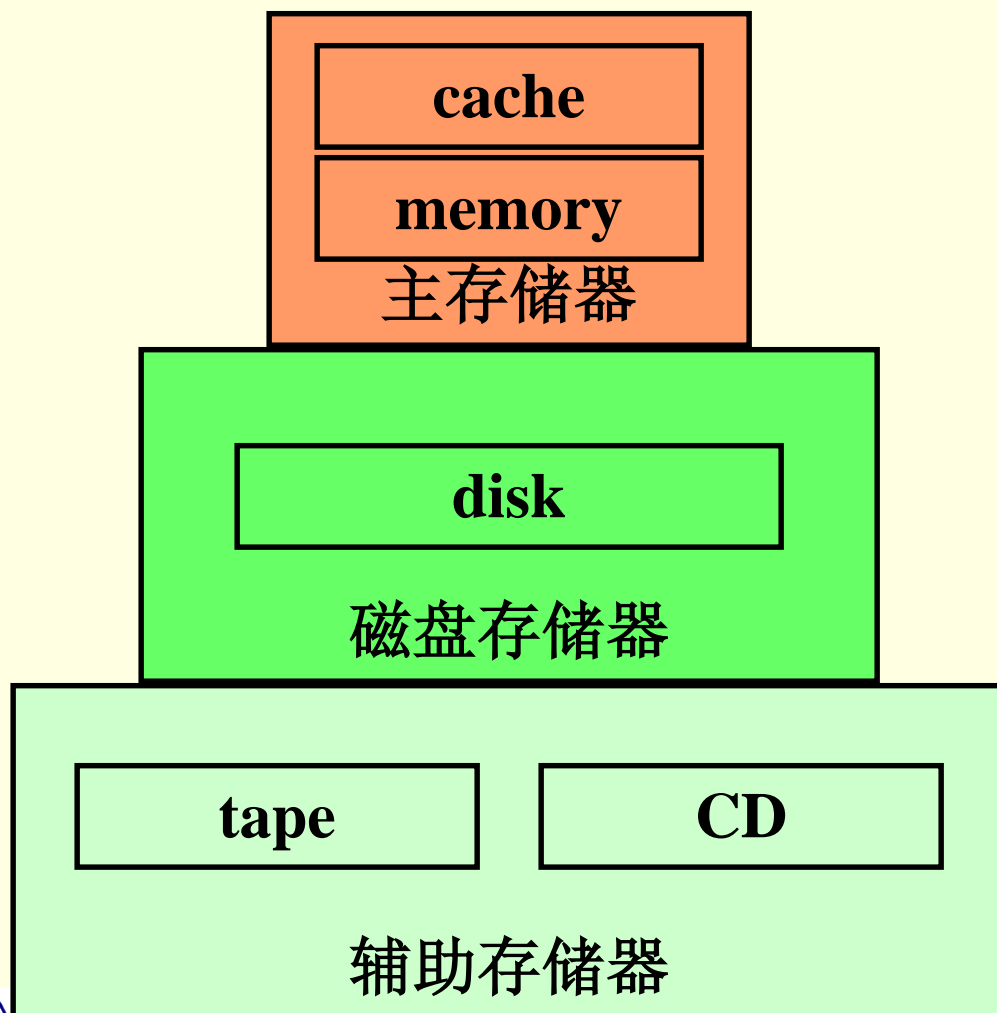


5.1 数据库存储结构

	存储容量	访问速度	访问类型	存取单位
第一级存储器	100MB ~ 10GB	10^{-8} 秒 ~ 10^{-7} 秒	随机	字节
第二级存储器	10GB ~ 10^3 GB	10毫秒 ~ 30 毫秒	随机	物理块
第三级存储器	10^6 GB	几秒钟 ~ 几分钟	顺序	数据块



5.1 数据库存储结构



小

快

高



大

存储
容量



慢

访问
速度



低

制造
成本



5.1 数据库存储结构

- 磁盘的I/O操作：首先根据给出的物理块地址定位，然后读/写指定磁盘块上的数据，其存取时间开销包括：
 - 活动头的移动时间—寻道时间
 - 磁盘片的旋转定位时间—等待时间
 - 读/写数据时间—传输时间
- 物理块是磁盘与内存进行数据交换的基本单位，因此物理块的大小是设计DBMS的重要参数。



5.1 数据库存储结构

- 磁盘的存取速度与内存的存取速度不匹配，为了有效地支持数据库的数据读写、提高数据库性能。为此，DBMS必须在内存开辟若干大小等于物理块的缓冲块或缓冲区 (Buffers)，并采用数据预取(Prefetching)和延时写(Delayed Writing)技术，减少 I/O操作。
- 即使外存（通常是硬盘）是非易失存储器，当系统（包括：OS、数据库系统本身、存储介质等）发生故障时，数据库不可避免地会遭受破坏，因此DBMS必须提供数据后备 (Backuping)功能。



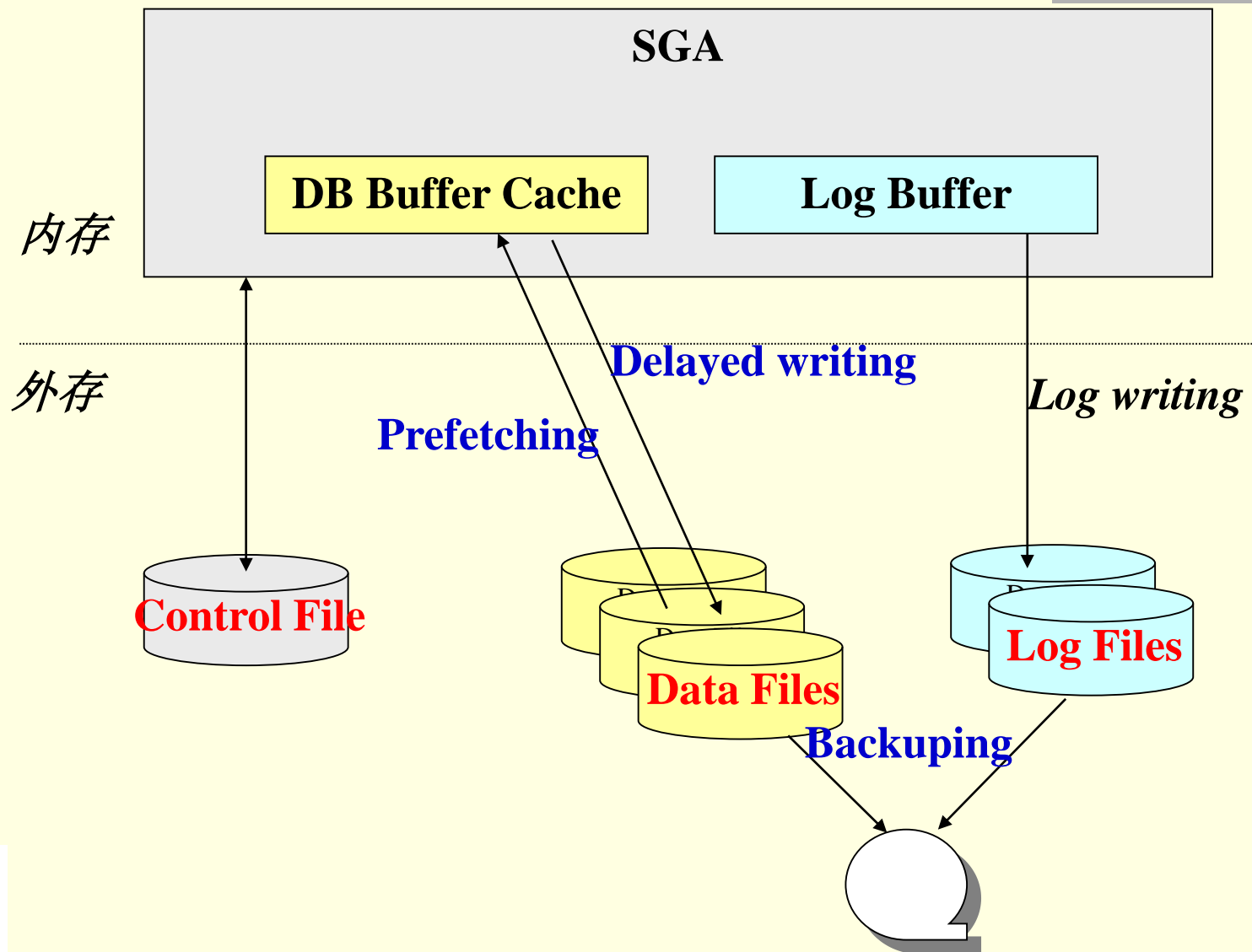
5.1 数据库存储结构

二、物理结构

- 数据库以多个文件(Files)的形式进行组织，并物理地存储于硬盘介质上。存储空间及文件由DBMS的存储管理器进行管理。(OS的存储管理和文件系统可为DBMS提供底层支持)。
- 通常，一个数据库有三种文件：
 - 数据文件(Data Files):
 - 用于存储数据库中的数据与元数据，一个数据库对应一个或多个数据文件。
 - 日志文件(Log Files):
 - 用于保存用户存取数据库的日志记录，一个数据库对应一个或多个日志文件。
 - 控制文件(Control Files):
 - 用于保存与数据库有关的若干参数(如：数据库名、数据库数据文件和日志文件的名字和位置，数据库的建立日期等)，一个数据库对应一个控制文件。



5.1 数据库存储结构



5.1 数据库存储结构

■ 三、逻辑结构

- 数据库用户并非直接与数据库的物理结构(物理存储介质或物理文件)打交道，DBMS的存储管理器提供了物理 \longleftrightarrow 逻辑的映像(Mapping)，使得用户直接面对数据库的逻辑结构。
- 逻辑结构涉及两个方面：
 - ①数据库的存储空间如何逻辑地划分与组织（即逻辑存储空间）；
 - ②用户如何使用数据库的数据（即用户模式及其对象）。



5.1 数据库存储结构

三、逻辑结构(cont.)

■ 逻辑存储空间：（以Oracle为例介绍）

- **表空间(Table Space)**：数据库的逻辑存储单位。一个数据库可包含一个或多个表空间；一个表空间可跨越多个磁盘分配。一般地，在数据库初始化时，系统总是自动建立一个缺省表空间(如Oracle中的SYSTEM表空间)，DBA事后可定义其他表空间。
- **段(Segment)**：表空间中一种指定类型的逻辑存储结构。有：
 - **数据段**：每个表/簇集有一个数据段，用于存储其中的数据。
 - **索引段**：每个索引有一个索引段，用于存储索引数据。
 - **回滚段**：由DBA建立，用于临时存储要回滚（撤消）的信息，以便事务回滚。
 - **临时段**：当一个SQL语句需要临时工作区时，由DBMS建立，用完后再回收。
- **范围(Extent)**：一个段由一组范围组成，范围是数据库存储空间分配的**逻辑单位**。
- **数据块(Data Block)**：一个范围由一组连续的数据块所组成，数据块是DBMS进行I/O的最小**物理单位**，其大小可不同于OS的标准I/O块大小。



5.1 数据库存储结构的特点

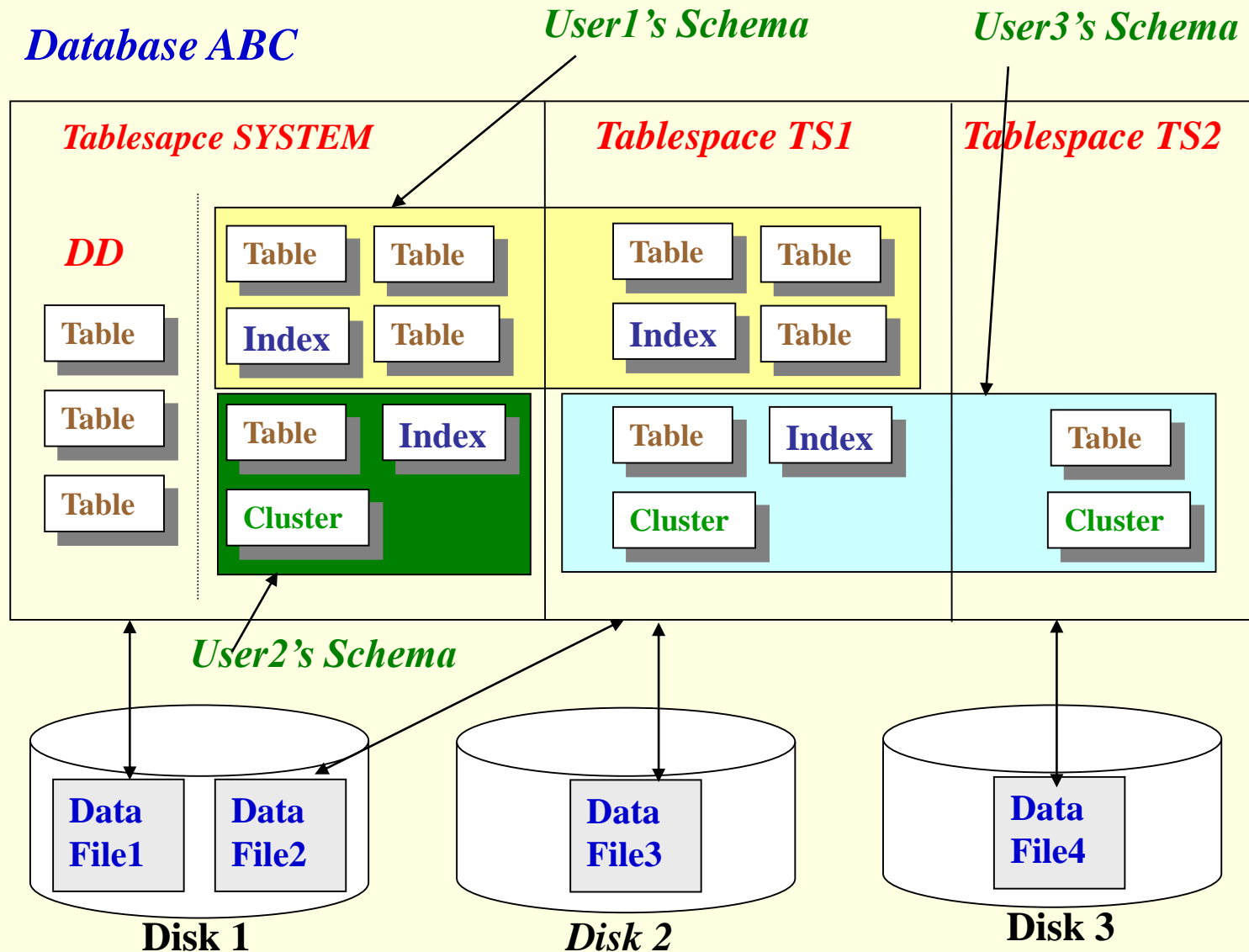
■ 三、逻辑结构(cont.)

■ 用户模式及其对象

- 模式(Schema): 每个数据库用户对应一个模式。
- 模式对象(Schema Objects): 一个模式中包含的数据逻辑结构对象。如：表，视图，索引，簇集，过程、触发器等。



5.1 数据库存储结构的特点



目录 Contents

■ 5.1 数据库存储结构

- 多级存储
- 物理结构
- 逻辑结构

■ 5.2 关系数据库中表的典型存储机制

- 索引
- 散列
- 簇集



5.2 关系数据库中表的典型存储机制一索引

- 在数据库中数据查找的速度是至关重要的，但是当数据库中数据量增大时，数据查找速度就会受到影响，当数据量极大时，查找速度将会受到严重影响，为解决此问题需引入索引、散列和簇集技术。
- **索引（Index）**是与表或簇集相关的一种可选存储机制，其通过一棵有序树（如B树）将索引键（Index Key, IK）值与数据块（的地址）建立联系，以提高数据检索性能。



5.2 关系数据库中表的典型存储机制一索引

一、顺序文件

- 按照某个属性(项)的取值进行排序而构成的数据文件被称为**顺序文件**。
- 例（假设每个物理块可以存放2条记录）

1	记录 1	}	第 1 个物理块
2	记录 2		
3	记录 3	}	第 2 个物理块
4	记录 4		
5	记录 5	}	第 3 个物理块
6	记录 6		
⋮	⋮		⋮

主键



5.2 关系数据库中表的典型存储机制一索引

[例1]

- 设一个关系有 10^6 个元组，在每个物理块(大小为4KB)中可存放10个这样的元组，则该关系大约占用：

10^5 个磁盘块（约400MB）

- 设该关系中的元组已经按照主键值从小到大的顺序构成了一个顺序文件，因此可以采用二分查找法来根据主键值进行记录定位。

- 按照一个主键的值进行记录定位所需要的磁盘I/O次数为：

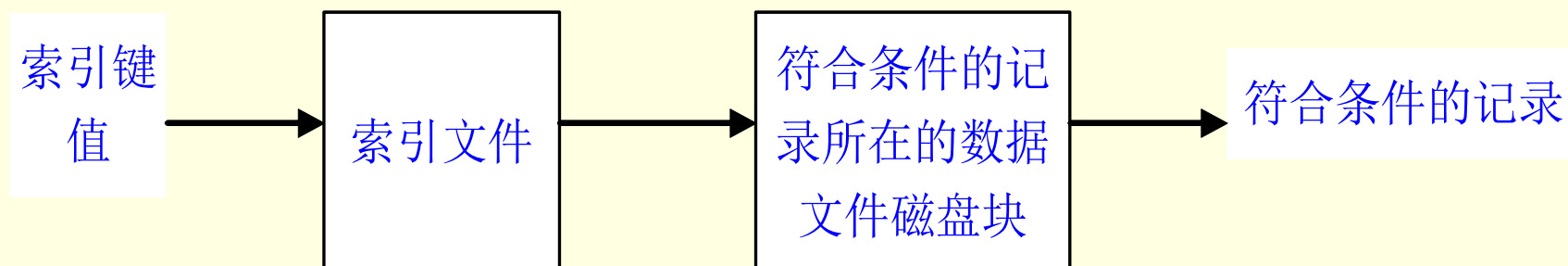
$$\log_2 10^5 \approx 16.6 \approx 17$$



5.2 关系数据库中表的典型存储机制一索引

二、索引文件

- **建立索引文件的目的：**以一个或多个字段的**索引键的值**为输入，能够快速找出具有该索引键值的记录。
- **利用索引文件进行记录查找的过程：**首先在索引文件中按照**索引键的值**进行查找，找出具有该索引键值的记录指针（即记录在数据文件中的存储地址），从而可以在数据文件中进行直接定位，读出所需要的记录。
- **通过建立索引文件可以大大提高在数据文件上的记录查找定位速度。**



利用索引文件访问数据文件中的记录的示意图



5.2 关系数据库中表的典型存储机制一索引

- 索引文件的大小一般都大大小于数据文件的大小。
 - 索引文件主要用于记录的快速定位操作，在索引文件中一般只含有索引属性上索引键(Index Key, IK)的值和记录的地址(指针)。如：

索引键值1	对应记录1的指针
索引键值2	对应记录2的指针
索引键值3	对应记录3的指针
.....

- 索引文件采用便于查找的特殊组织结构(如B或B⁺树)。
- 在不同类型的数据文件上可以建立不同的索引文件。



5.2 关系数据库中表的典型存储机制一索引

- 索引键如是PK或UNIQUE列，称**主索引(Primary Index)**，否则称**次索引(Secondary Index)**。
- 主索引由于没有两行在索引键上具有重复值，故也称**唯一索引(Unique Index)**。
- 一般地，唯一索引由DBMS自动建立，而非唯一索引则需由DBA/用户自己显式地建立。
- 索引键可以是列组，此时称**组合索引(Composite Index)**。
- 若每个索引键值均有对应的索引项(Index Entry)，称这样的索引结构为**稠密索引(Dense Index)**，否则称**非稠密索引**。



5.2 关系数据库中表的典型存储机制一索引

三、在顺序文件上的索引

- 稠密索引
- 非稠密索引
- 多级索引
 - 在讨论上述的三种索引文件时，我们假设其索引键值具有唯一性（主索引）。



5.2 关系数据库中表的典型存储机制一索引

■ 稠密索引(dense index)

■ 索引文件

- 存放记录的索引键值以及指向记录本身的指针(记录的存储地址), 并且按照索引键值的顺序进行排序。
- 一个索引键值和一个记录指针构成的键-指针对, 称为一个索引项:

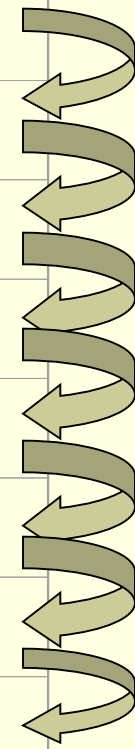
索引键值K	记录指针P
-------	-------

■ 稠密索引

- 若每个索引键值均有对应的索引项(Index Entry), 称这样的索引结构为稠密索引。
- 例如:



5.2 关系数据库中表的典型存储机制一索引

S#	指针	S#	SN	SD	SA	
S ₁	● →	S ₁	LU	CS	18	
S ₂	● →	S ₂	LI	CS	17	
S ₃	● →	S ₃	XU	MA	18	
S ₄	● →	S ₄	LO	CS	18	
S ₅	● →	S ₅	LIN	PH	19	
S ₆	● →	S ₆	WANG	CS	17	
S ₇	● →	S ₇	SEN	MA	17	
S ₈	● →	S ₈	EHEN	PH	18	

稠密索引

数据文件



5.2 关系数据库中表的典型存储机制一索引

- 利用稠密索引进行数据查找要比直接在数据文件上进行数据查找的速度快。其原因是：
 - 索引文件使用的物理块比数据文件的少，因此磁盘I/O的时间开销小。
 - 一般情况下，一个物理块中可以存放的索引项的个数要远远大于可以存放的记录数。
 - 索引文件中的索引项被按照索引键值进行了排序，因此在索引文件中可采用二分查找法来提高查找速度。
 - 这在无序的数据文件(堆文件)中是无法做到的。
 - 索引文件可能足够小，可以放在内存中操作，从而不必访问磁盘。



5.2 关系数据库中表的典型存储机制一索引

- 利用稠密索引查找关键字值为K的记录算法如下：
 - 采用二分查找法在索引文件中查找是否存在索引键值为K的索引项；
 - 如果不存在相关的索引项，则表示主键值为K的记录不存在，查找失败。否则：
 - 根据找到的索引项中的记录指针到数据文件中进行直接定位，读取相应的记录。



5.2 关系数据库中表的典型存储机制一索引

[例2]

- 为[例1]中的顺序数据文件建立一个稠密索引。假设每个磁盘块可以存放100个索引项，则该稠密索引共有 10^6 个索引项，需要占用：

10^4 个磁盘块（约40MB）

- 利用该稠密索引进行记录定位需要的磁盘I/O次数为：

$$\log_2 10^4 + 1 \approx 13.3 + 1 \approx 15$$

- 其中的1是访问数据文件的磁盘I/O。



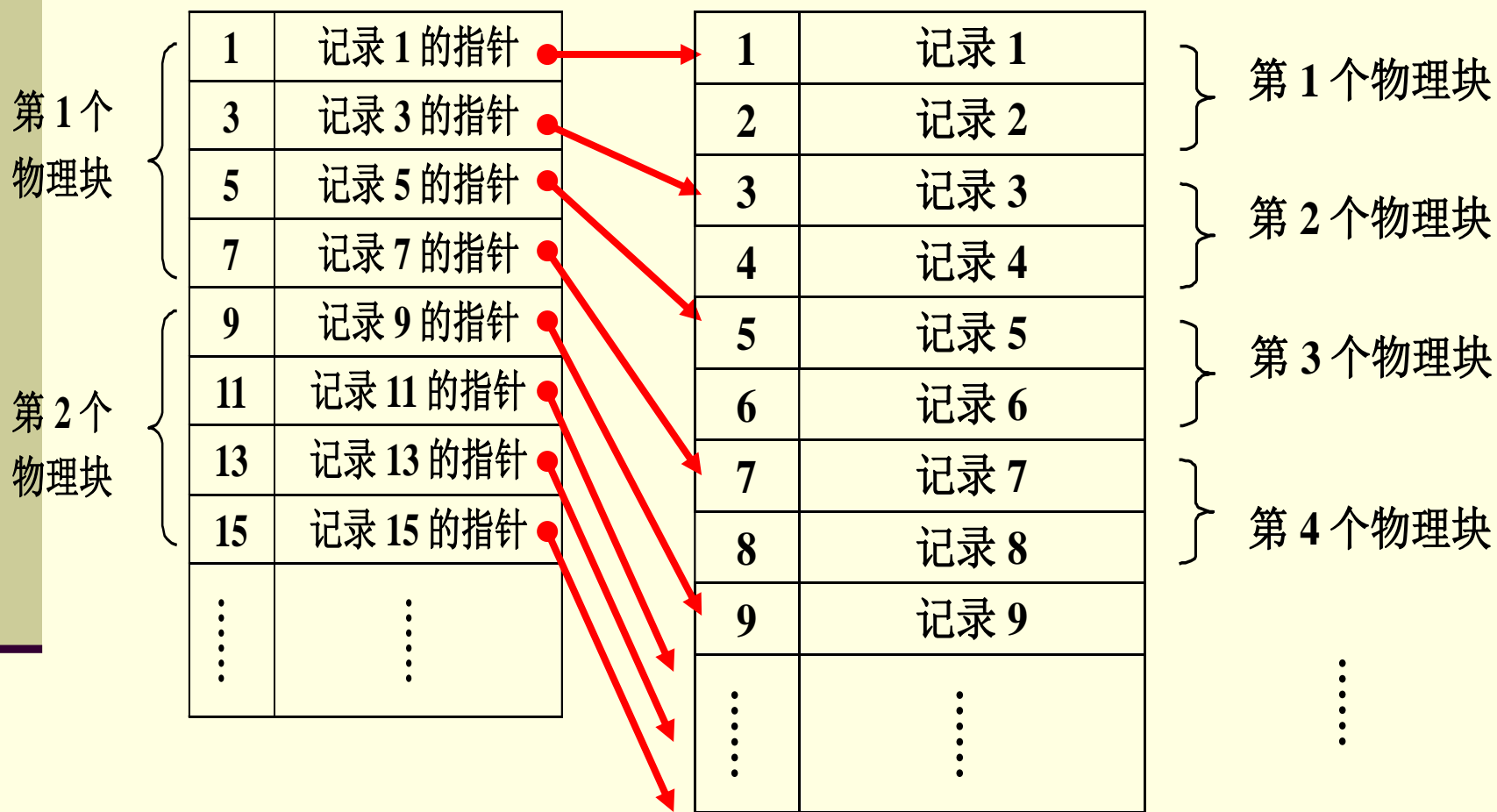
5.2 关系数据库中表的典型存储机制一索引

■ 非稠密索引(nondense index)

- 如果数据文件是顺序文件，我们可以在索引文件中只为数据文件的每个物理块设一个索引项，记录该物理块中第一条数据记录的索引键值及该物理块的首地址。
- 这样建立起来的索引文件被称为非稠密索引。



5.2 关系数据库中表的典型存储机制一索引



非稠密索引

数据文件



5.2 关系数据库中表的典型存储机制一索引

- 利用非稠密索引查找关键字值为K的记录的算法如下：
 - 采用二分查找法在索引文件中查找索引键值小于或等于K，且最接近K的索引项；
 - 如果不存在相关的索引项，则表示主键值为K的记录不存在(所有记录的索引键值均大于K)，查找失败。否则：
 - 根据找到的索引项中的记录指针到数据文件中读取相应的物理块 D；
 - 在物理块 D 中利用二分查找法查找主键值为K的记录。
 - 数据文件是顺序文件。



5.2 关系数据库中表的典型存储机制一索引

■ [例3]

- 为例1中的顺序数据文件再建立一个非稠密索引。
由于该数据文件共占用 10^5 个磁盘块，因此非稠密索引文件中有 10^5 个索引项，需要占用：
 - 10^3 个磁盘块（约4MB）
- 利用该非稠密索引进行记录定位需要的磁盘I/O次数为：
 - $\log_2 10^3 + 1 \approx 9.97 + 1 \approx 11$



5.2 关系数据库中表的典型存储机制一索引

■ 稠密索引与稀疏索引的区别

■ 索引文件的定义不同

- 非稠密索引只能用于顺序文件上的索引组织。
- 稠密索引中的每个索引项对应数据文件中的一条记录，而非稠密索引中的每个索引项则对应数据文件中的一个物理块。

■ 需要的磁盘空间大小不同

■ 在记录的查找定位功能上存在差别：

- 稠密索引：可以直接回答是否存在键值为K的记录。
- 非稠密索引：需要额外的磁盘I/O操作，即需要将相应的数据文件中的磁盘块读入内存后才能判别该记录是否存在。



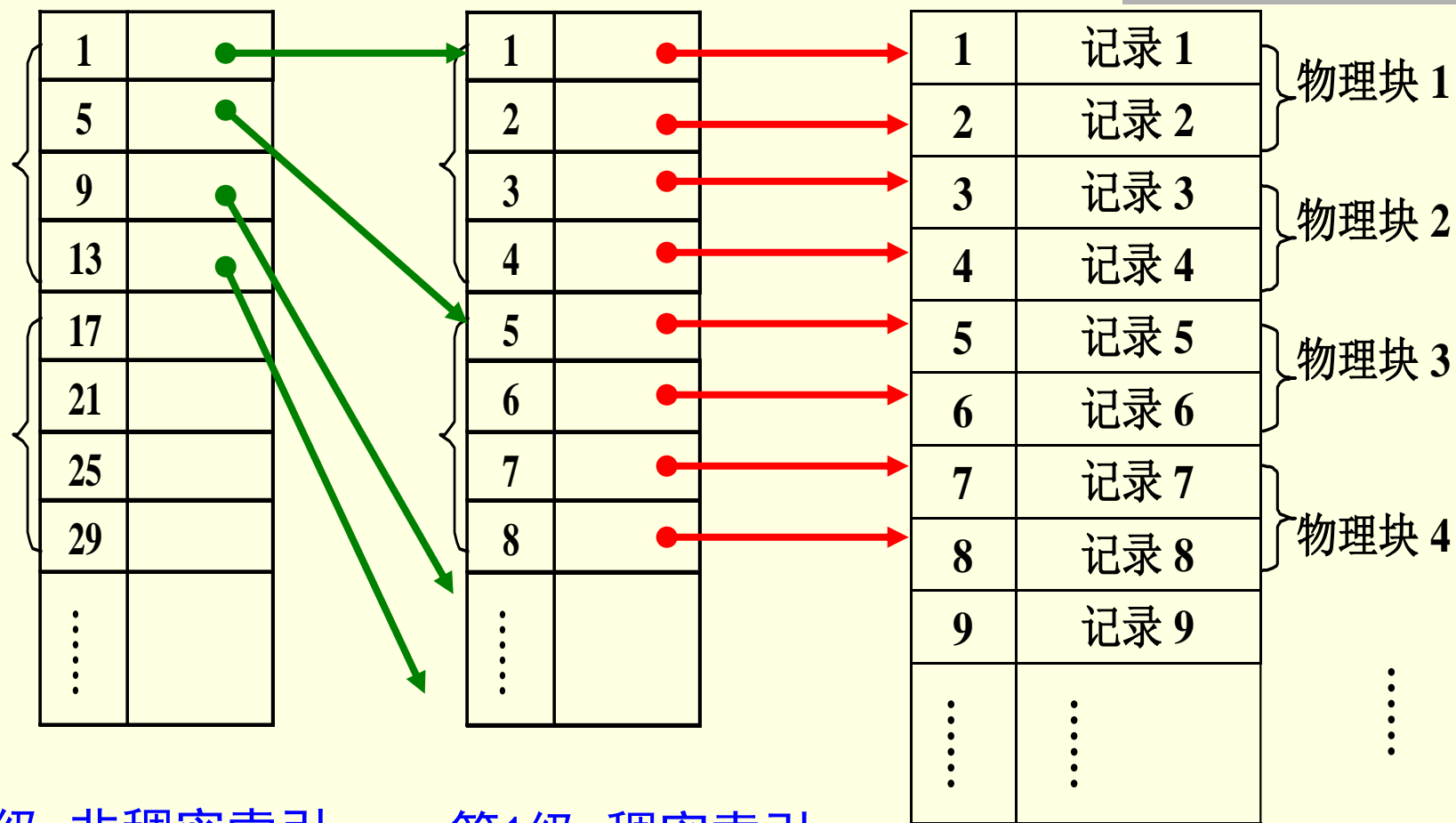
5.2 关系数据库中表的典型存储机制一索引

■ 多级索引

- 索引文件本身也可能占据多个存储块，为了能够快速找到这些索引块在磁盘中的存储位置，需要引入新的索引结构，即在索引文件上再建立索引，从而构成了多级索引。
 - 由于索引文件本身是顺序文件，因此索引文件上的索引结构采用的是非稠密索引。
- 将直接建立在数据文件上的索引（例2、例3中建立的索引）称为第一级索引，根据第一级索引文件建立的索引称为第二级索引，依此类推，从而可以建立一个多级索引结构。
 - 在多级索引组织结构中，第一级索引可以是稠密索引，也可以是非稠密索引。从第二级索引开始建立的都是非稠密索引。



5.2 关系数据库中表的典型存储机制—索引



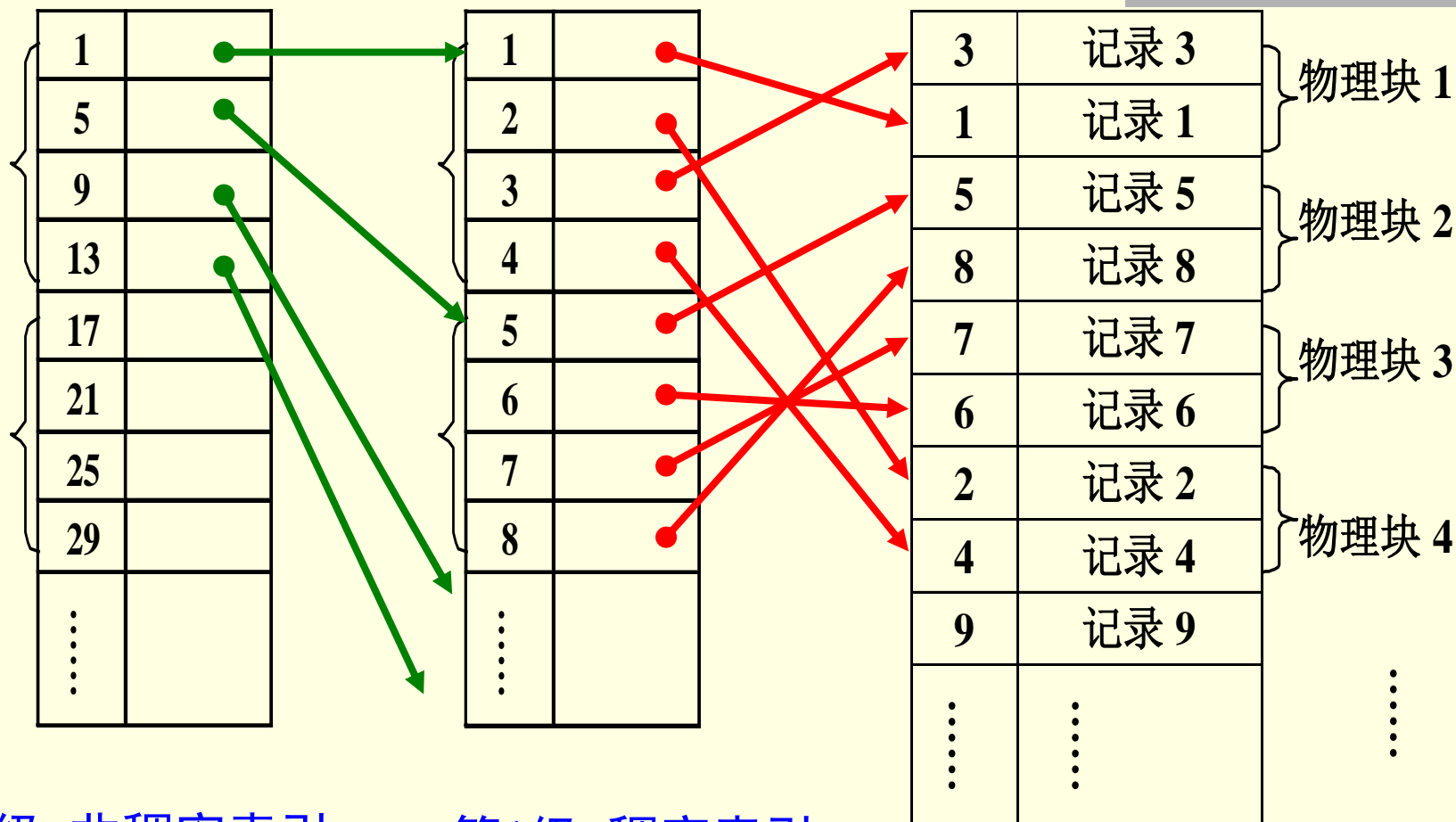
第2级 非稠密索引

第1级 稠密索引

数据文件 (顺序文件)



5.2 关系数据库中表的典型存储机制一索引



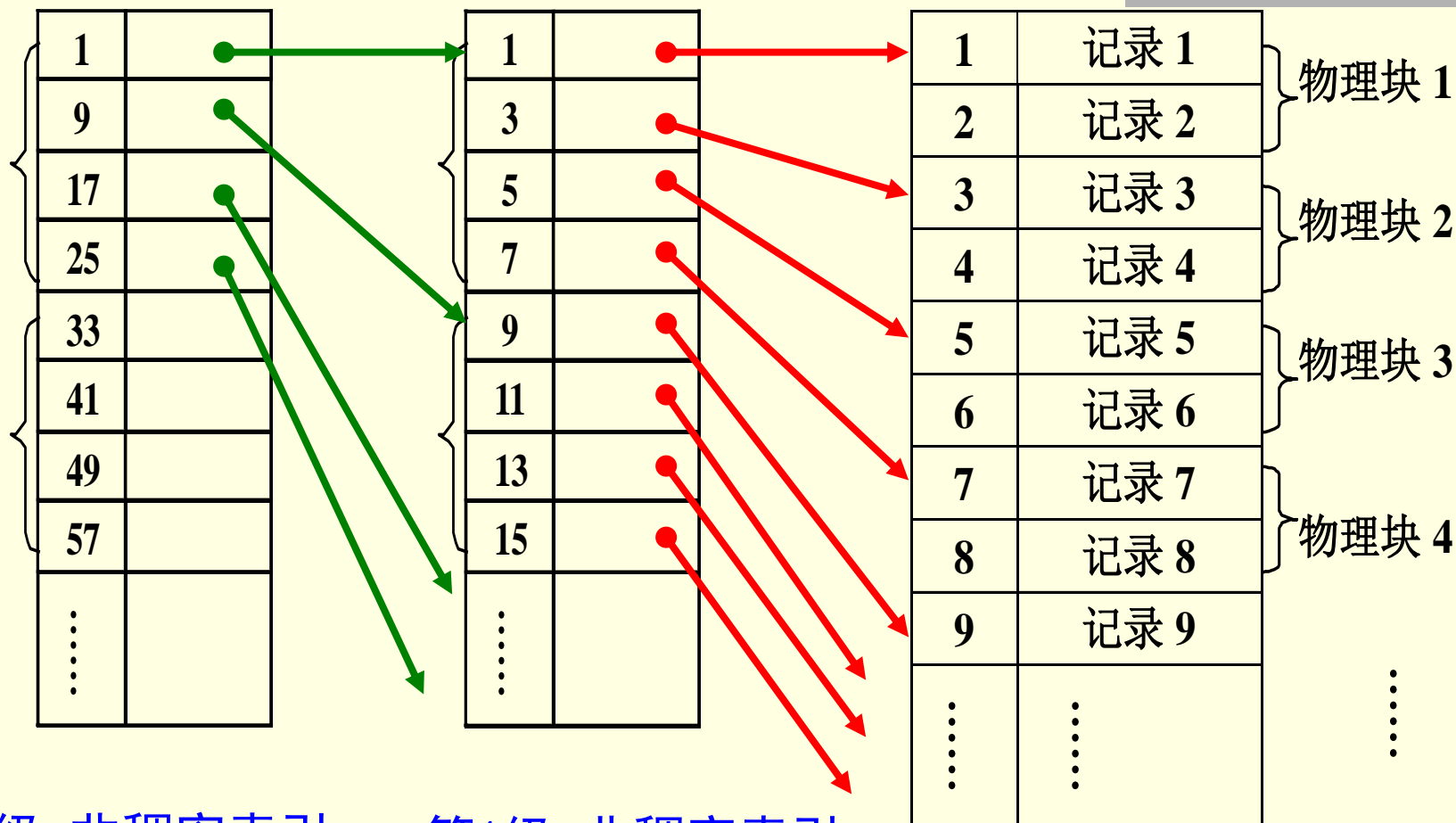
第2级 非稠密索引

第1级 稠密索引

数据文件 (堆文件)



5.2 关系数据库中表的典型存储机制—索引



第2级 非稠密索引

第1级 非稠密索引

数据文件 (顺序文件)



5.2 关系数据库中表的典型存储机制一索引

■ [例4]

- 在例2中建立的稠密索引文件上再建立一个非稠密索引。由于例2中的稠密索引文件共占用 10^4 个磁盘块，因此新建的非稠密索引文件中有 10^4 个索引项，需要占用

100个物理块（约400KB）

■ [例5]

- 在例3中建立的非稠密索引文件上也可以再建立一个非稠密索引。由于例3中的非稠密索引文件共占用 10^3 个磁盘块，因此新建的非稠密索引文件中有 10^3 个索引项，只需要占用：

10个物理块（约40KB）



5.2 关系数据库中表的典型存储机制一索引

- 考虑在例4和例5中所建立的第二级索引文件，由于它们只分别占用400KB和40KB的存储空间，这样的索引文件完全可以全部放在内存中。
- 因此，在这样的索引文件上进行搜索定位不需要索引文件上的磁盘I/O操作，我们只要根据在二级索引中查找到的索引项到第一级索引文件中直接读取相应的索引磁盘块，并根据在该物理块中所找到的索引项到数据文件中直接读取相应的数据文件物理块或记录。因此，根据这样的两级索引结构进行记录的查找定位只需要2次磁盘I/O操作，大大低于我们在例2和例3中所需的磁盘I/O次数（见下表）。



5.2 关系数据库中表的典型存储机制一索引

	索引类型	需要的磁盘空间 (包括数据文件和索引文件)	记录查找需要的磁盘I/O 次数
例 1	无索引，直接在顺序数据文件上进行记录的查找	100000	17
例 2	一级稠密索引	110000	15
例 3	一级非稠密索引	101000	11
例 4	基于一级稠密索引的二级索引	110100	2
例 5	基于一级非稠密索引的二级索引	101010	2



5.2 关系数据库中表的典型存储机制一索引

四、非顺序文件中的索引结构

- 在数据库系统中所使用的数据文件通常都是无序的（堆文件组织），因此更加需要利用上述的索引结构来建立非顺序文件上的索引，以提高记录查找的速度。
- 如果索引键值在非顺序数据文件中具有唯一性，则可以按照下述步骤建立其上的索引文件：
 - 首先为非顺序数据文件建立第一级的稠密索引；
 - 然后再根据需要建立该稠密索引上的多级非稠密索引。

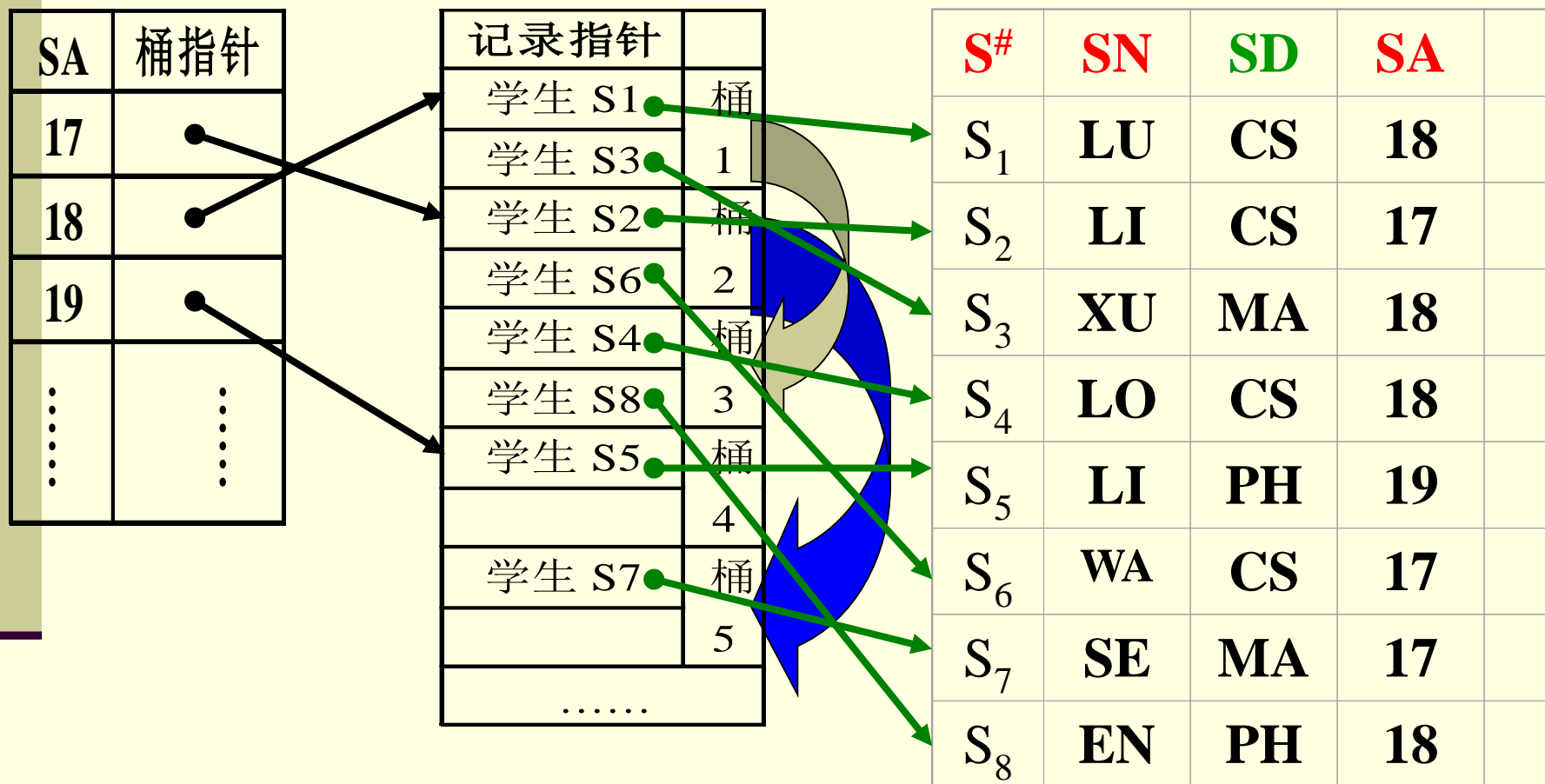


5.2 关系数据库中表的典型存储机制—索引

- 在次索引中，索引键值不唯一，则可以通过在第一级的稠密索引和数据文件之间加一个记录指针桶(bucket)。此时索引项(K_i, P_i)中的记录指针 P_i 不再是指向数据文件中的记录，而是指向一个记录指针桶，在桶中存放着索引键值为 K_i 的记录的记录指针。
 - 在这里，记录指针桶的大小 BSize 是预先确定的。
 - 当在数据文件中插入第一条索引键值为 K_i 的记录 R_i 时，在索引文件中将生成一个新的索引项(K_i, P_i)，同时系统将自动申请一个大小为 BSize 的记录指针桶 B_i ，将指针 P_i 指向该记录指针桶 B_i ，同时将当前记录 R_i 的记录指针保存在记录指针桶 B_i 中。
 - 当新的索引键值为 K_i 的记录被加入到数据文件中时，只需要将新记录的记录指针加入到记录指针桶 B_i 中。如果记录指针桶 B_i 已满，系统将申请一个新的记录指针桶，并链接到原来的记录指针桶的后面。



5.2 关系数据库中表的典型存储机制一索引



稠密索引

记录指针桶

索引键值不唯一，且数据文件没有按照索引键值排序存储



5.2 关系数据库中表的典型存储机制—索引

- 索引文件本身是一个顺序文件，利用索引文件或多级索引可以大大提高数据文件的访问效率。
- 索引顺序文件的不足
 - 记录查找算法的效率不高 ($\log_2 N$) ；
 - 在数据文件是非顺序文件，或索引键值的变化比较频繁的情况下，索引顺序文件自身的维护非常复杂，对索引项的插入、修改和删除操作会导致索引项在索引文件中的大量移动。
 - 在上述情况下，如果通过引入链接磁盘块的方法来减少索引项的移动，又会减低存储空间的利用率，并最终影响到系统的性能。
 - 因此需要引入适合于索引文件的存储组织的文件结构，这就是在数据库系统中广泛使用的多级索引技术：B/B⁺树索引



5.2 关系数据库中表的典型存储机制—索引

■ 五、B-tree

- B树(B-tree)是一种动态平衡多分树(Dynamic Balanced Multiway Tree)。B/B⁺树是一种动态、多级索引组织方法，是适合于组织存放在外存的大型磁盘文件的一种树状索引结构。其中用得比较多的是B⁺树。
- B/B⁺树的结点划分
 - 叶结点：B/B⁺树的最下一级索引是树的叶结点
 - 内部结点：B/B⁺树中的其它结点(非叶结点)，其中：
 - 根结点：B/B⁺树的最上一级索引是树的根结点
- 在B/B⁺树中，由叶结点所构成的最下面的一级索引总采用稠密索引，而其它层次上的索引则采用非稠密索引。
- 运用B树的索引结构总是动态索引。B树的维护（即索引的维护）由DBMS进行，对用户是透明的。



5.2 关系数据库中表的典型存储机制一索引

■ B⁺树中的结点

- 每个结点占用一个物理块，每个结点能容纳 n 个键和 $n+1$ 个指针，将 n 取得尽可能的大，以便在一个物理块中存放更多的索引项。
- 例如：设每个物理块的大小是4096个字节，每个键值占4个字节，每个指针占8个字节，则满足： $4n + 8(n+1) \leq 4096$ 的最大 n 的值是340。
- B⁺树的结点的结构如下：

P_1	K_1	P_2	K_2	P_m	K_m	P_{m+1}
-------	-------	-------	-------	-------	-------	-------	-----------



5.2 关系数据库中表的典型存储机制—索引

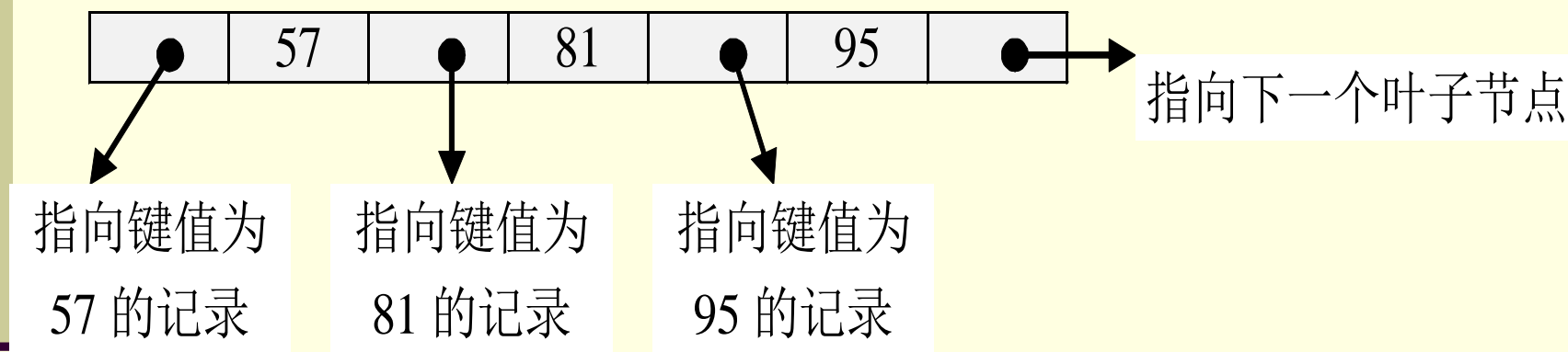
P_1	K_1	P_2	K_2	P_m	K_m	P_{m+1}
-------	-------	-------	-------	-------	-------	-------	-----------

- 其中： K_1, K_2, \dots, K_m 是索引关键字值，且 $K_1 < K_2 < \dots < K_m$
 - ❖ 若是叶子结点，则： $\lfloor (n+1)/2 \rfloor \leq m \leq n$ ， P_1, P_2, \dots, P_m 是指向数据记录的指针， P_{m+1} 是指向其右边的下一个叶子结点的指针。其中， P_i 是指向关键字值为 K_i 的数据记录（ $i=1, 2, \dots, m$ ）。
 - ❖ 若是根结点，则 $1 \leq m \leq n$ ，
 - ❖ 否则(即内部结点)： $\lceil (n-1)/2 \rceil \leq m \leq n$
其中： $P_1, P_2, \dots, P_m, P_{m+1}$ 分别指向另一棵子树的根结点。
 - ❖ 若是非叶结点，则：
对 P_1 所指向的子树中的任意一个索引键 K ，都有 $K < K_1$
对 P_{m+1} 所指向的子树中的任意一个索引键 K ，都有 $K \geq K_m$
对 P_j 所指向的子树中的任意一个索引键 K ，都有 $K_{j-1} \leq K < K_j$



5.2 关系数据库中表的典型存储机制—索引

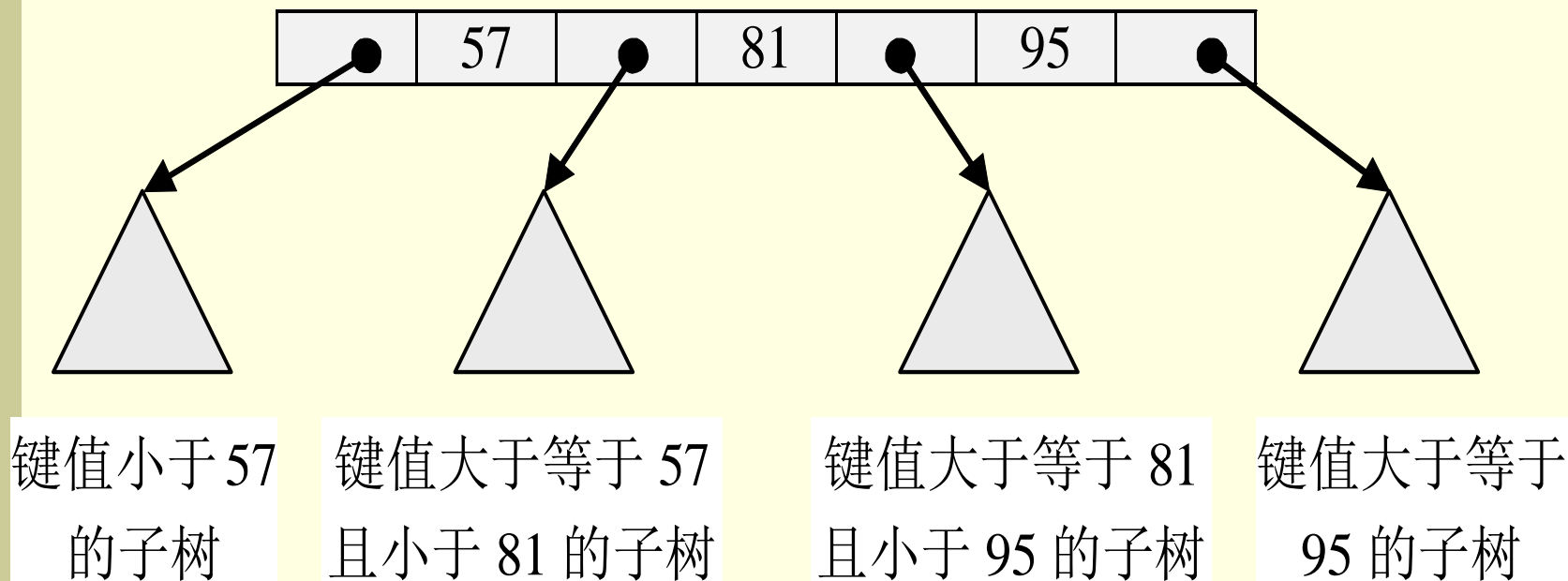
- 例如：有一棵秩 $n=3$ 的B⁺树，则每个结点最多可放3个关键字和4个指针，每个叶子结点至少有2个关键字和3个指针，每个内部结点至少有1个关键字和2个指针。



秩为 3 的 B⁺树的某个叶子节点



5.2 关系数据库中表的典型存储机制一索引



秩为 3 的 B⁺树的某个内部节点



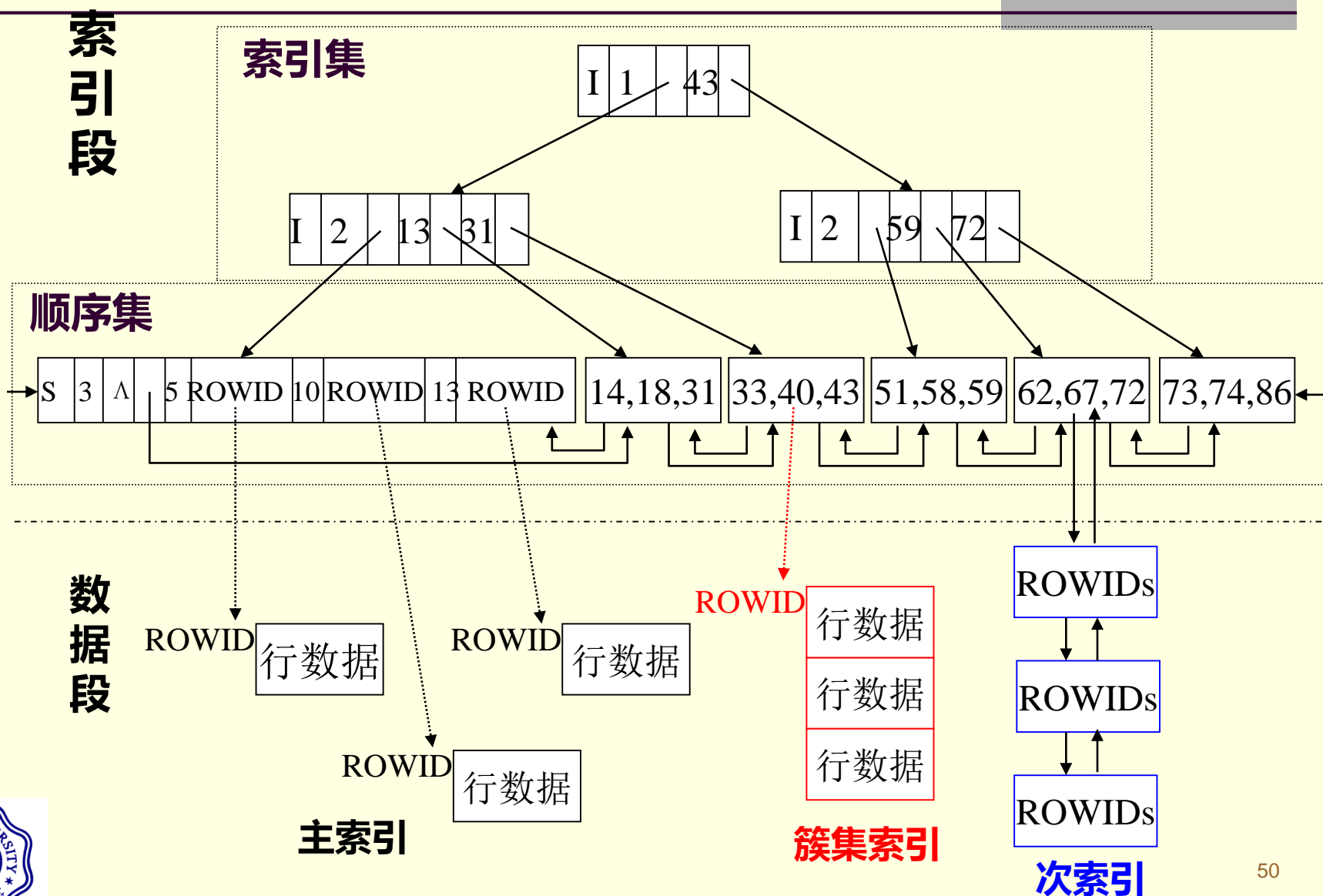
5.2 关系数据库中表的典型存储机制—索引

[例]: 设索引键值集为: {5, 10, 13, 14, 18, 31, 33, 40, 43, 51, 58, 59, 62, 67, 72, 73, 74, 86}

则Oracle B*树的索引结构示意图为:



5.2 关系数据库中表的典型存储机制一索引



5.2 关系数据库中表的典型存储机制—索引

■ 利弊

■ 利

- 当表的容量较大时（需占较多数据块时），索引能明显提高数据查询性能。
- ——最理想的块I/O次数为：B树深度+1。
- 索引特别适合于指定行查询和范围查找。

■ 弊

- 增加了系统维护的开销，特别是当表的数据需频繁更新时。



目录 Contents

■ 5.1 数据库存储结构

- 多级存储
- 物理结构
- 逻辑结构

■ 5.2 关系数据库中表的典型存储机制

- 索引
- 散列
- 簇集



5.2 关系数据库中表的典型存储机制—散列

- **散列 (Hash)** 是与表或簇集相关的一种可选存储机制，由于可通过一个Hash函数将**散列键 (Hash Key)** 的值映射成一个数据块的地址，给出散列键的值 K_i 立即可通过 $h(K_i)$ 得到其对应的存储物理块地址，从而可明显改进数据检索的性能。

数据文件

...	
...	
...	
...	
...	
K_i	
...	
...	

$h(K_i)$

Hash Key的值 K_i

.....



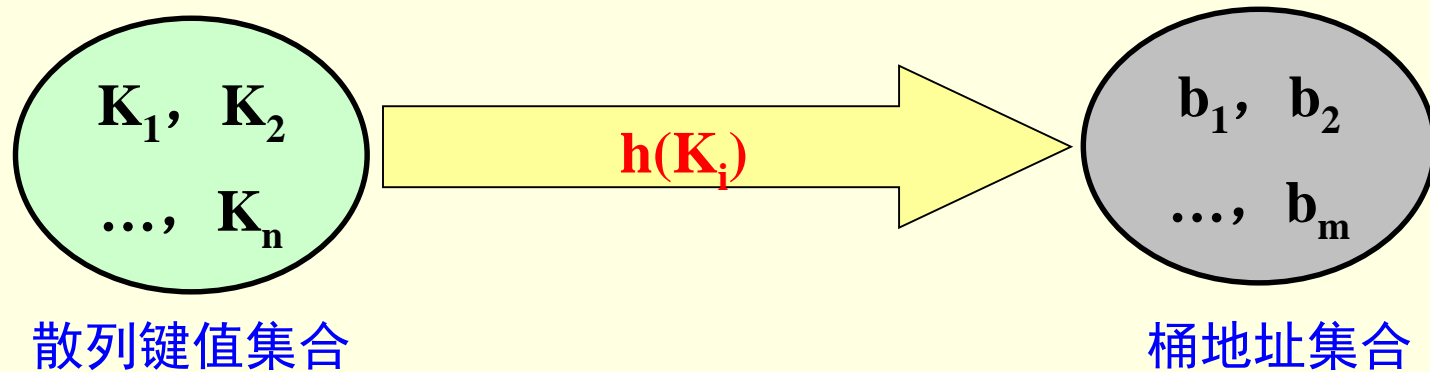
5.2 关系数据库中表的典型存储机制—散列

■ 静态散列的实现方法

- 建立数据文件的散列键 K 以及该键值的集合 $\{ K_1, K_2, \dots, K_n \}$
- 建立磁盘物理存储单位桶(Bucket)以及桶地址的集合 $\{ b_1, b_2, \dots, b_m \}$ 。
 - 一个桶可以存放多条记录（或记录指针）
 - 一个桶可以是一个磁盘物理块，也可以是比较磁盘块还大的物理空间。
- 设计散列函数 $\text{hash}(K_i)$
 - 以建立数据文件中散列键的值与桶（桶地址）之间的对应关系，即一个 K_i 通过 $\text{hash}(K_i)$ 必能找到唯一的一个桶地址。
 - 设计良好的散列函数将使得 n 个指定项值被平均分配到 m 个桶中去。



5.2 关系数据库中表的典型存储机制—散列



- 在散列技术中，桶的空间大小是固定的，即一个桶中可以存放的记录（指针）数是固定的。
- 但是在实际应用中，记录在指定项值上的分布往往是不均衡的，从而使得在某些桶中存在空间浪费现象，而另外一些桶则存在空间溢出问题。
- 当一个桶的空间溢出时，需要通过链接的方法申请“溢出桶”与其相联，以达到扩大桶空间的目的。

5.2 关系数据库中表的典型存储机制—散列

■ 静态散列技术

■ 优点

- 按Hash键值访问数据，速度快

■ 缺点

- Hash键值映射的地址空间有限；
- 用Hash键值寻址时，同一个Hash键值可能对应多个记录，不同的Hash键值可能映射到同一个地址。
- 只对Hash键值到记录的访问有效，对其他类型的访问不一定有效
- 处理变长记录不便
- 很难找到通用的Hash函数
- 当桶（Bucket）装满后的溢出处理较为复杂等原因，在数据经常变动的数据库环境中不宜使用，故使用动态散列（桶的数目可以分动态变化；桶可分裂/合并）较多。



5.2 关系数据库中表的典型存储机制—散列

- 在理想下，基于散列键的单行查询只需一次I/O即可。但散列带来好处的情况并不多，例如：Oracle就如此，以至于把“散列”就说成是“散列簇集”。
- 散列簇集(Hash Cluster)，即对簇表的行在簇集键列上运用Hash函数进行散列，以决定相应数据块的物理地址。
- Oracle提供缺省的Hash函数，支持单列/列组上的散列簇集。用户也可以自己提供Hash函数，但此时有限制：簇集键/散列键必须由单列组成，且只允许取整数值。



5.2 关系数据库中表的典型存储机制—散列

■ 利弊

■ 利

- 若Hash值对每行是唯一的，此时使用散列簇集最为理想。
——只需1次块I/O。

■ 弊

- 若Cluster Key或Hash(Cluster Key)有许多相同的值，则用散列簇集并不好。
——因为地址冲突必须将数据块链到溢出表，会降低存取速度。



目录 Contents

■ 5.1 数据库存储结构

- 多级存储
- 物理结构
- 逻辑结构

■ 5.2 关系数据库中表的典型存储机制

- 索引
- 散列
- 簇集



5.2 关系数据库中表的典型存储机制—簇集

■ 机制

- 簇集(Cluster)是存储表数据的一种可选方法。
- 一个簇集是一个/组表，这个/组表中具有同一公共列(组)值的所有行均存储在一起(即物理上同一或相邻的数据块中)。这些公共列(组)称簇集键(Cluster Key, CK)。



5.2 关系数据库中表的典型存储机制—簇集

- **[例]:** 表emp与dept均有deptno列, 可将deptno列作为簇集键创建一个簇集, 将两表数据一起存储在该簇集中。

CREATE CLUSTER personnel (department_number INT) ; /*创建簇集*/

```
CREATE TABLE emp
(empno INT PRIMARY KEY,
ename VARCHAR(12) NOT NULL,
...
deptno INT NOT NULL)
CLUSTER personnel (deptno) ; /* 称emp为 (已) 簇表 */
```

```
CREATE TABLE emp
(deptno INT,
dname VARCHAR(10),
loc VARCHAR(12))
CLUSTER personnel (deptno) ; /* 称dept为 (已) 簇表 */
```



5.2 关系数据库中表的典型存储机制—簇集

■ 簇集有两种实现方法

■ 索引簇集 (Indexed Cluster)

- 对簇表在簇集键上再建索引，每个簇集键值有一个索引项。(缺省)
- [例]：建立索引簇集personnel：

```
CREATE CLUSTER personnel (department_number INT) INDEX ;
```

■ 散列簇集 (Hash Cluster)

- 对簇表的行在簇集键列上运用Hash函数进行散列，以决定相应数据块的物理地址。这样，具有同一散列值的行将存储在一起。
- [例]：建立散列簇集personnel：

```
CREATE CLUSTER personnel (department_number INT)  
HASH IS department_number HASHKEYS 100 ;
```



5.2 关系数据库中表的典型存储机制—簇集

■ 利弊

■ 利

- 可改进簇表间在簇集键列上连接运算的性能。
——因为减少了磁盘I/O次数。
- 节省存储空间。
——因为簇表中每个簇集键值只存储一次，不管这个/些表中有多少行包含此簇集键值。

■ 弊

- 降低了簇表上更新运算(INSERT, UPDATE, DELETE)的性能。
——因为增加了系统维护开销。

