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

第八章　关系数据库存储管理

Principles of Database Systems

第八章 关系数据库存储管理

8.1 数据库存储

8.2 缓存

8.1.1 数据存储概述

8.1.2 数据组织

文件、页、元组、表、日志

8.1.3 索引结构

顺序表索引、辅助索引、B+树、哈希索引、位图索引

8.1.4 系统目录

8.1.5 存储模型　NSM、DSM

8.1.1数据存储概述

2类DBMS：

- 面向内存（memory-oriented）的DBMS
- 面向磁盘（disk-oriented）的DBMS
  - DBMS假定其数据存储在非易失磁盘上。
  - DBMS的若干组件负责数据在易失内存和非易失磁盘间传送。

- 问题：存储介质的选择。
  - 磁盘、内存，易失/非易失的问题
  - 存储技术发展：“非易失性内存”

查询计划

操作执行

存取访问方法

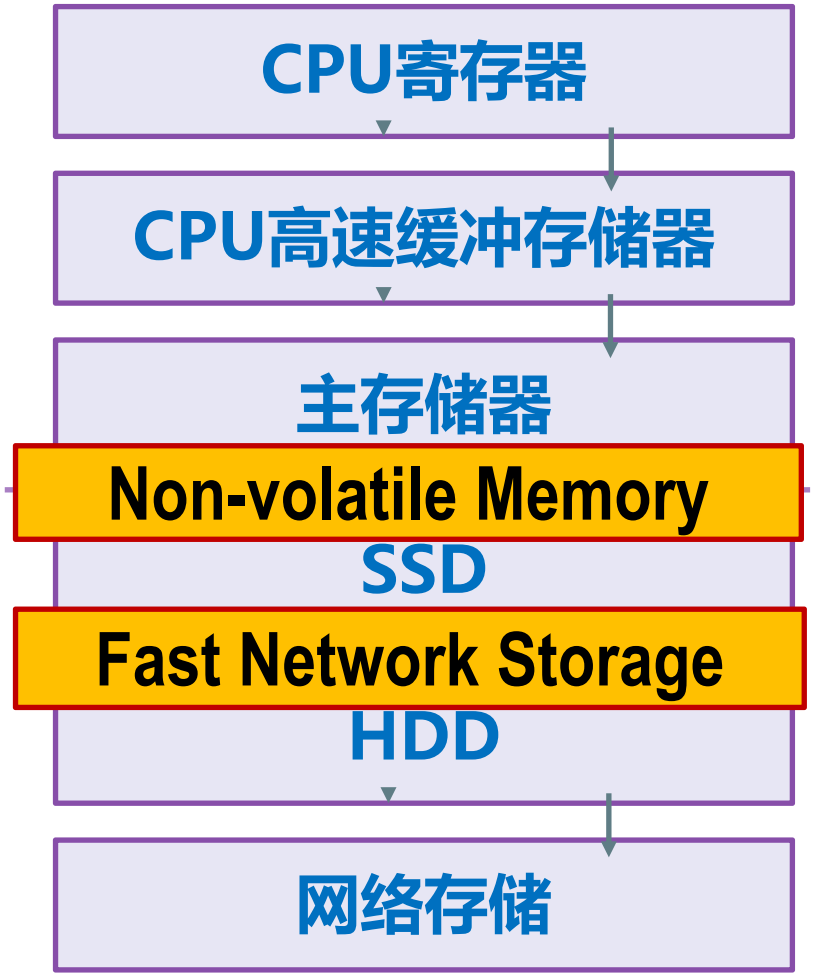
缓冲池管理器

磁盘管理器

用于数据库的存储介质及其架构

- **易失**
- 随机访问
- 可按字节寻址

- **非易失**
- 顺序访问
- 可按“块”寻址



访问速度快
容量小
昂贵

访问速度慢
容量大
便宜

介质访问时间

0.5 ns	L1 Cache Ref
7 ns	L2 Cache Ref
100 ns	DRAM
150,000 ns	SSD
10,000,000 ns	HDD
~30,000,000 ns	Network Storage
1,000,000,000 ns	Tape Archives

磁盘和主存储器之间数据传输的单位为“块”。

本节主要关注如何隐藏磁盘的延迟问题。

数据库存储管理目标

目标：

1. “最优数据组织”：存储效率高；存取效率高！
2. 磁盘型DB：允许DBMS管理可超过内存大小的数据库。
3. 由于读/写磁盘代价很高，存储管理要能避免大的延时和性能下降。
4. 由于磁盘随机访问比顺序访问慢得多，DBMS希望能“最大化”顺序访问。

- 顺序访问（Sequential Access）

  连续的请求通常处于相同或相邻的磁道上连续的块，因此只有第一块需要“磁盘寻道”，后续不需要。

- 随机访问（Random Access）

  一个数据集所在的块可能散布在整个存储空间，每一次请求都需要“磁盘寻道”，其效率低于顺序访问模式。

磁盘块访问的优化

I/O操作代价较高，DBMS领域为了提高访问块的速度，形成了很多技术：

- 缓冲（Buffering）
- 预读（Read Ahead）
- 调度（Scheduling）（电梯算法）
- 文件组织（File Organization）（数据临近存放，相邻柱面，同区，重组）
- 非易失性写缓冲区（Nonvolatile Write Buffer）（Raid控制器常用）
- 日志磁盘（Log Disk）（减少写等待时间）
- 其他：多磁盘、磁盘镜像、 RAID

面向磁盘的DBMS

特征：

- 数据库文件存储在磁盘上，数据被组织成“页“，第一页是目录页。
- 缓冲池管理磁盘和内存间数据交换：磁盘I/O对性能影响巨大。

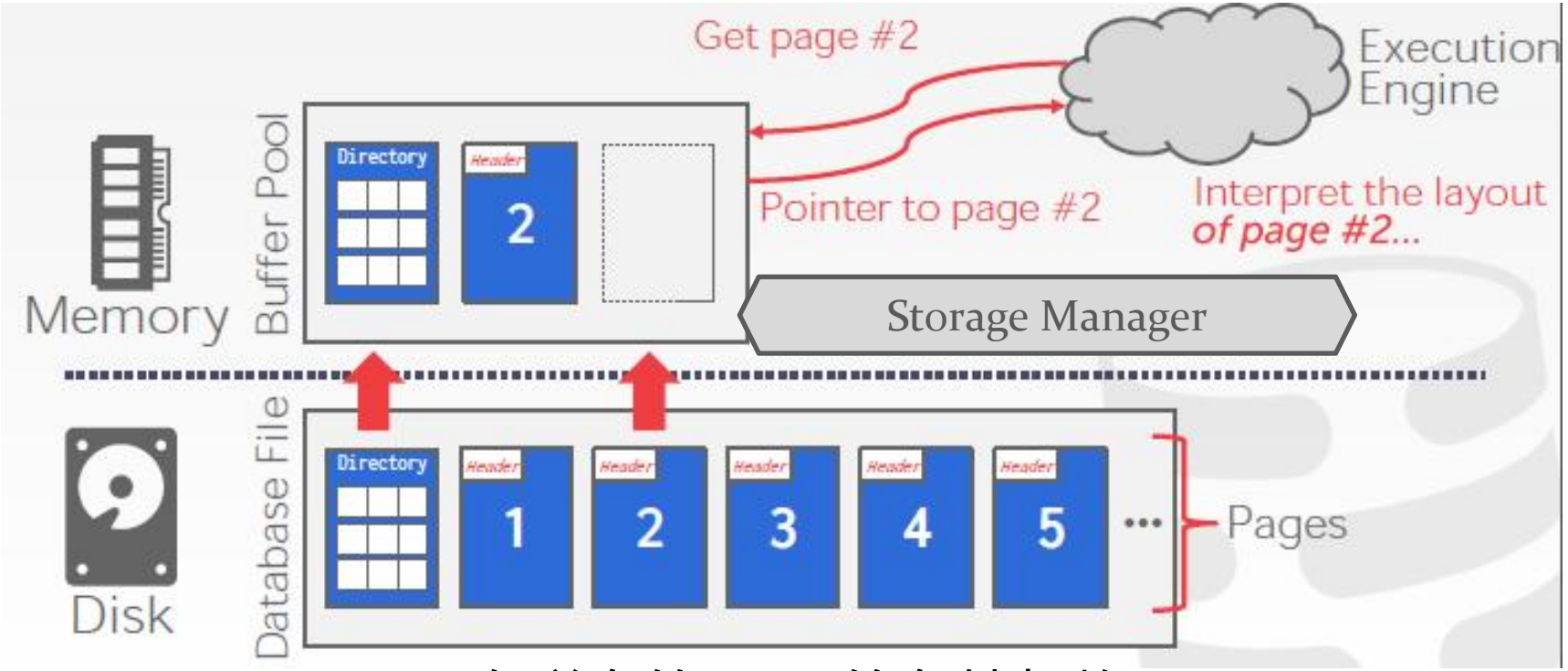
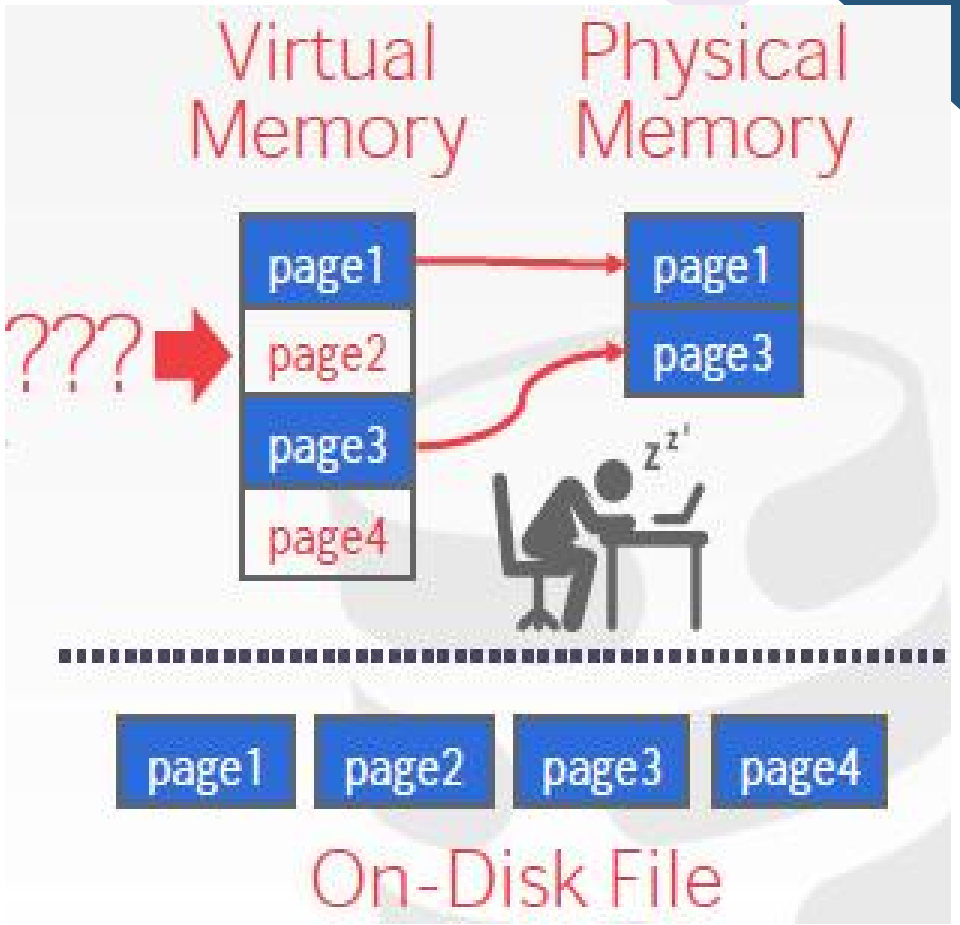


图 面向磁盘的DBMS的存储架构

面向磁盘的DBMS VS. OS

- 上例中，能否使用OS替代DBMS?
- 一种常规思路是内存映射机制mmap，即将文件内容映射到进程地址空间，再建立进程的虚拟地址空间Page到物理内存中Page的映射。
- 由OS负责在文件页和内存间数据交互。
- 如果是“读”操作，是可以胜任的，例如当出现“缺页”时，进程“阻塞”，可能可接受
  - 多线程访问mmap文件



但凡需要磁盘空间代替内存时，mmap都可以发挥功效。

面向磁盘的DBMS VS. OS

- 如果是“写”操作呢?
- 由于日志、并发控制等实现的需要，OS并不知道哪些page需要在其他page之前写到磁盘。
- 可能的解决方法是引导OS的页缓存替换机制：
  - madvise：告知OS何时计划读取特定页或内存使用模式；
  - mlock：告知OS某内存范围不能被替换出去；
  - msync：告知OS某内存范围被刷新到磁盘。

MonetDB和早期的MongoDB如此。

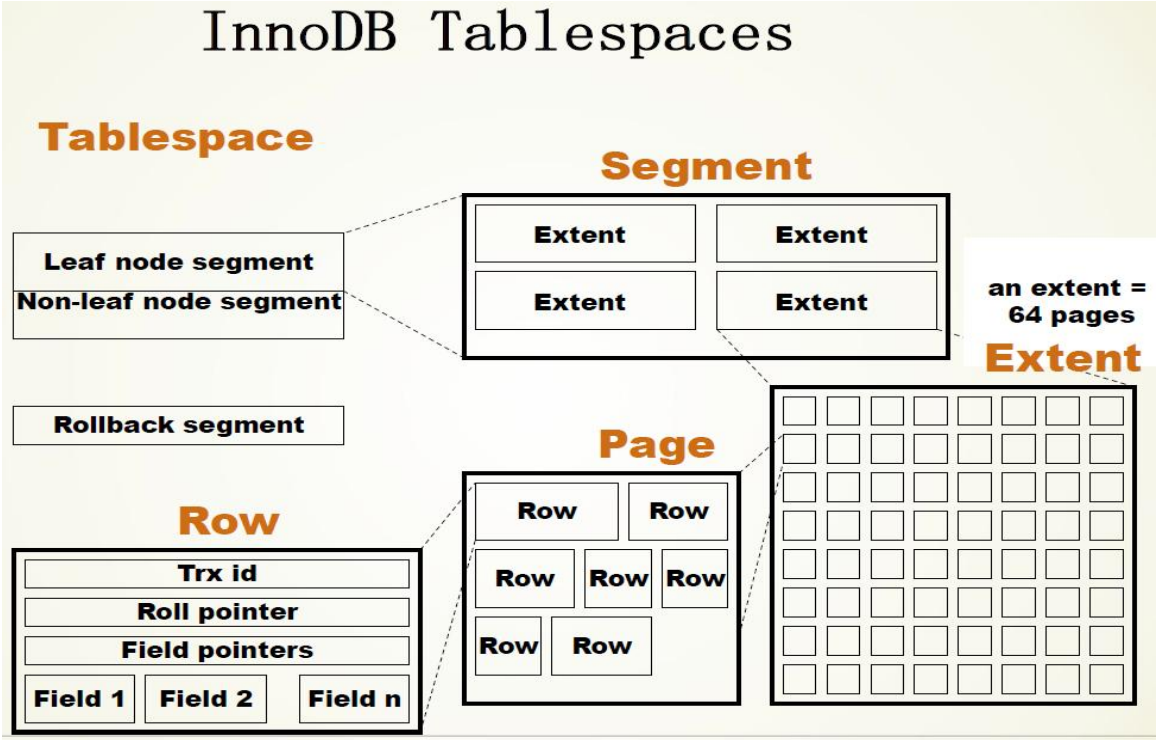
从效率、安全等角度出发，主流DBMS都倾向于自己来进行页面的管理，可更好的支持：

- **按正确顺序将“脏”页刷新到磁盘；**
- **更为可靠的数据预读取；**
- **缓冲区替换策略；**
- **线程/进程调度。**

数据库存储面临的主要问题

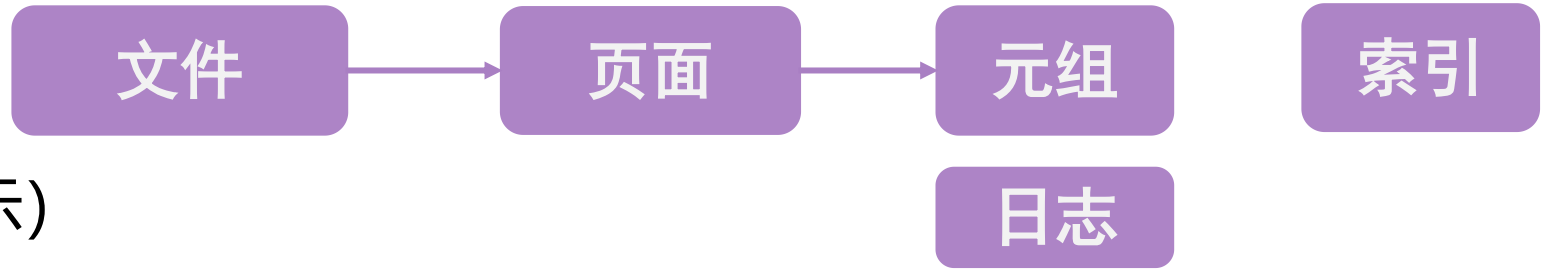
- 问题1：DB如何 逻辑表示 / 磁盘存储 数据库文件?
  - 数据库逻辑组织方式和物理存储结构
- 问题2：DBMS如何管理数据在内存和磁盘间的交互?
  - 缓冲池设计

- File Storage
- Page Layout
- Tuple Layout



8.1.2 数据组织

1. 数据库的物理组织方式与逻辑组织方式
2. 文件设计
3. 页设计
4. 元组表示（记录表示）
5. 关系表的组织
6. 日志式文件组织



8.1.2.1 数据库的物理组织方式与逻辑组织方式

- 数据库的物理组织的基本问题就是如何利用操作系统的基本文件组织来设计数据库的数据存放方法，实际也就对应了数据库存储管理的两种方式：

  （1）对象-文件：每个数据对象（基本表、索引等）都对应一个OS文件，本质上是将存储管理交由OS完成，如PG，KingBase等；

  （2）数据库-文件：整个数据库对应一个或若干个文件，DBMS进行存储管理，该方式称为段页式存储，如Oracle、SQL Server等。

  - 在段页式管理中通常先一次性申请一个大文件，然后随着数据的增多则增加新的文件。对文件内部，通常对数据库空间进行逻辑划分，常见划分方式：表空间-段-分区-数据块。

一种典型的数据库逻辑组织方式

- 常见的表空间：
系统表空间、联机表空间、临时表空间……

- 一个表空间通常包含一个或多个物理文件。
- 一个物理文件只能属于一个表空间。

- 一个段可以逻辑上组织不同类型的数据，如数据段、索引段、临时段等。

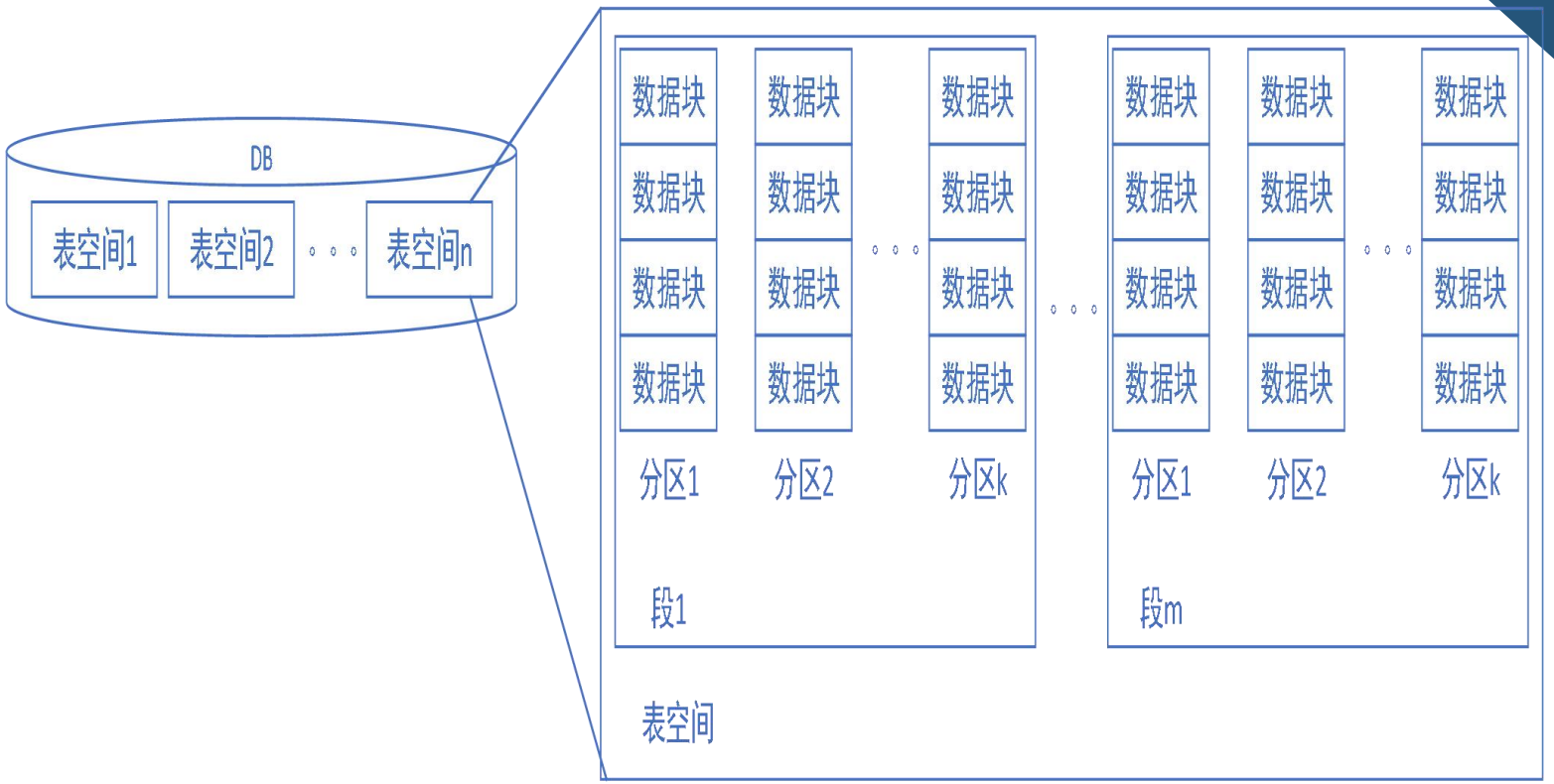


图　一种DB逻辑组织方式
表空间-段-分区-数据块

物理组织

- 从物理上看，数据库的数据则是以文件为单位组织，文件在物理上又划分为文件块，一个关系会占用一到多个物理块，而一个块存放关系的多个记录，即“文件 - 块 - 记录”的物理组织形式。
- 数据库的逻辑组织方式与物理组织方式的对应关系：

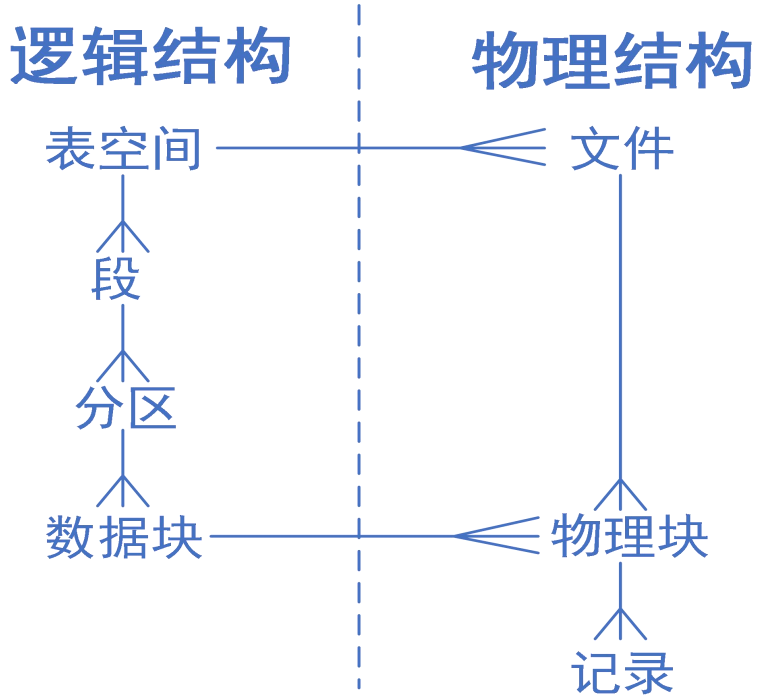


图　DB逻辑组织方式和物理组织方式的对应关系

8.1.2.2 文件设计

- 文件存储（File Storage）
  - DBMS通常按一定的自有、专有格式组织并将数据库存储在一个或多个磁盘文件中。

    OS并不知晓这些文件的组织形式和内容。
  - 早期DBMS（1980年代）在裸存储设备上使用自定义的文件系统，某些大型企业级DBMS依然支持该方式，后续大多数DBMS都不这么做。

- DBMS的“存储管理器”：负责数据库文件的管理，将文件组织为“页”的集合，追踪页面数据的读写操作，追踪可用的存储空间。

  通过对读/写操作的合理调度，提升页面访问的空间和时间局部性（效率）。

8.1.2.2 文件设计

- 页设计（Page Layout）
  - 数据库的页是一个固定大小（如4KB）的数据块。
  - 页可以容纳：
    - 不同类型数据：元组、元数据、索引、日志记录等。
    - 数据一般不混合存放，即一个页只存放一类信息（比如元组）。
    - 一些DBMS要求页面是“自包含(self-contained)”的。
  - 页ID：每个页具备一个唯一ID：
    - 当数据库只有单文件时，页面ID可以就是文件中的偏移地址。
    - 当有多个文件时，大部分DMBS会有个间接层来映射页面ID到文件的路径和偏移地址，系统上层访问页面号时，存储管理器将其转换为文件路径和偏移地址。

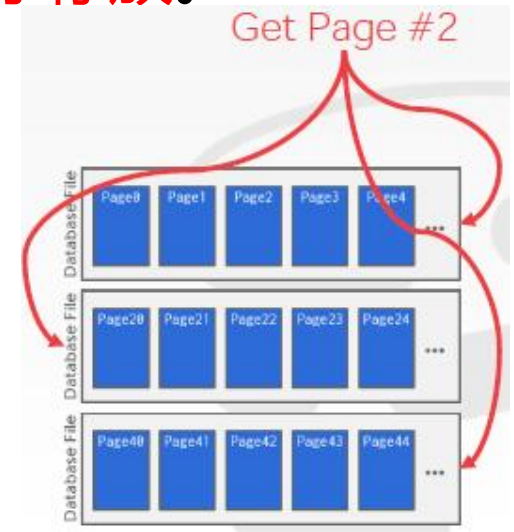
页的大小（Size of Page）

- 注意区分2种“页”：
  - 硬件页面（4KB）：
  - 数据库页面（512B-16KB）：通常是磁盘块大小的整数倍，是DBMS在磁盘和缓冲池间交换数据的基本单位。

- 硬件页面是存储设备中能保证故障安全写操作（failsafe write）的最大数据块单位，原子写。

页的堆文件组织方式

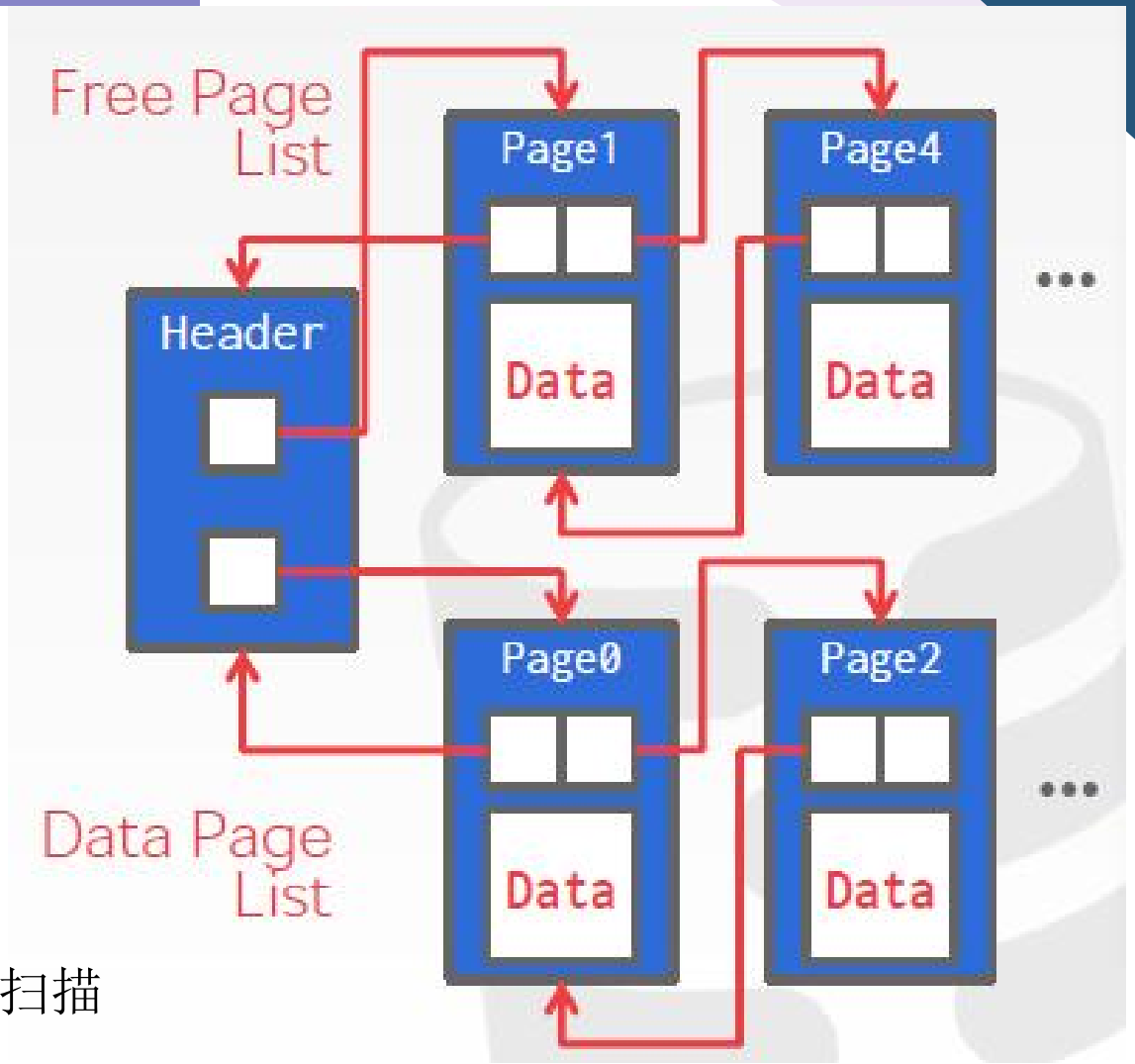
- 关系是记录的集合，这些记录在数据库文件中可以有3种组织方式：
  - 堆文件组织（Heap File Organization）
  - 顺序文件组织（Sequential File Organization）
  - 散列文件组织（Hash File Organization）
- Heap文件是一个无序的page集合，其中的元组可按随机顺序存放。
  - 支持page的创建、读、写和删除操作
  - 支持遍历所有page的操作
- 堆文件的2种表示方式：
  - 链表（Linked List）
  - 页目录（Page directory）



多文件时，需要元数据记录文件中有哪些页面，以及哪些页有空闲空间。

页的堆文件组织：链表

- 链表（Linked List）：以链表的形式将文件中的空闲页和数据页分别勾连起来。
- 堆文件头部设立一个header page，并存放两个指针，分别指向：
  - 空页列表（free page list）头部
  - 数据页列表（data page list）头部
- 每个page均记录当前空闲的空槽（slot）

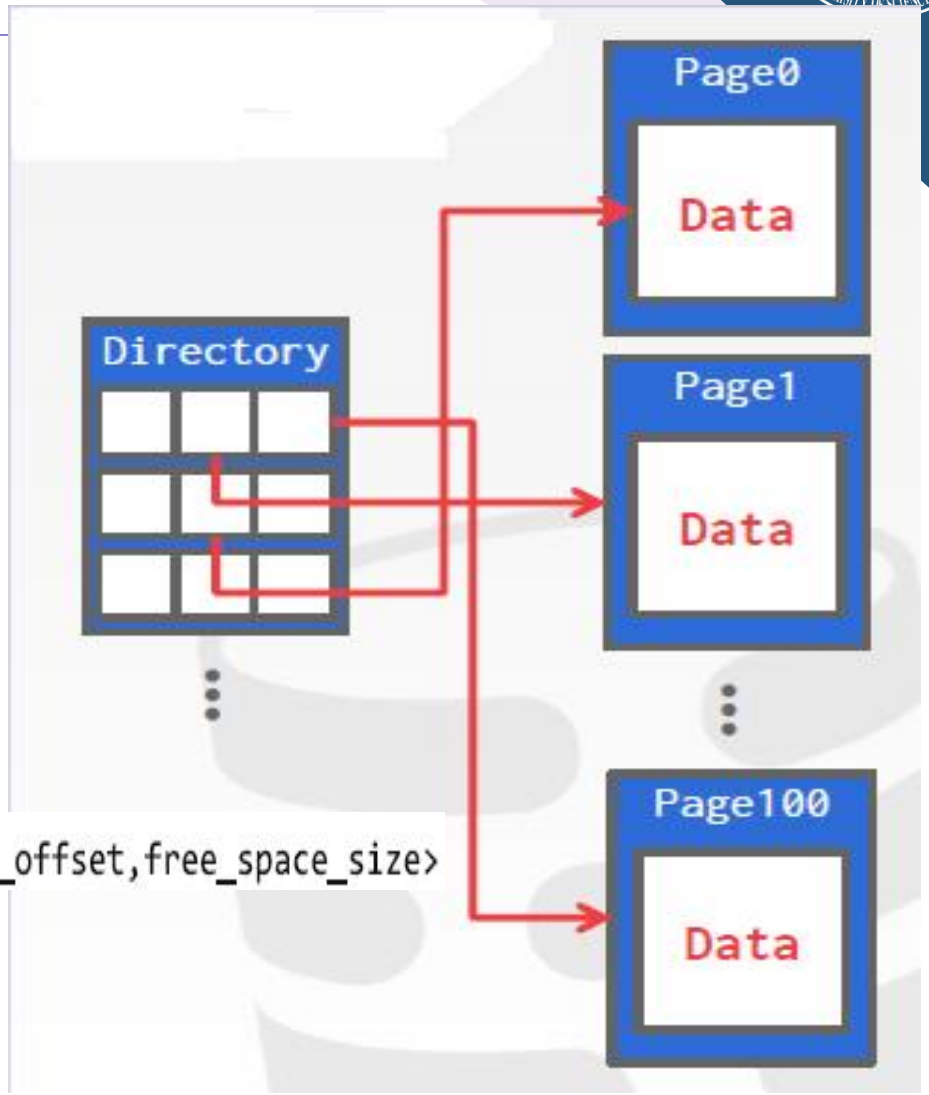


如要找一个特定数据页，需要从链首开始逐个扫描链表中的页面，直到找到为止，I/O开销较大。

页的堆文件组织：页目录

- 页目录（Page Directory）：维护一种特殊的页面（目录页），用于记录所有的数据页的存放位置。
- 该目录也同时记录每个页面的空槽信息。
- 页目录将页面的状态信息集中存放在一起，可提高查找特定页面的速度。

- DBMS必须保持目录页与所有页的当前信息同步。



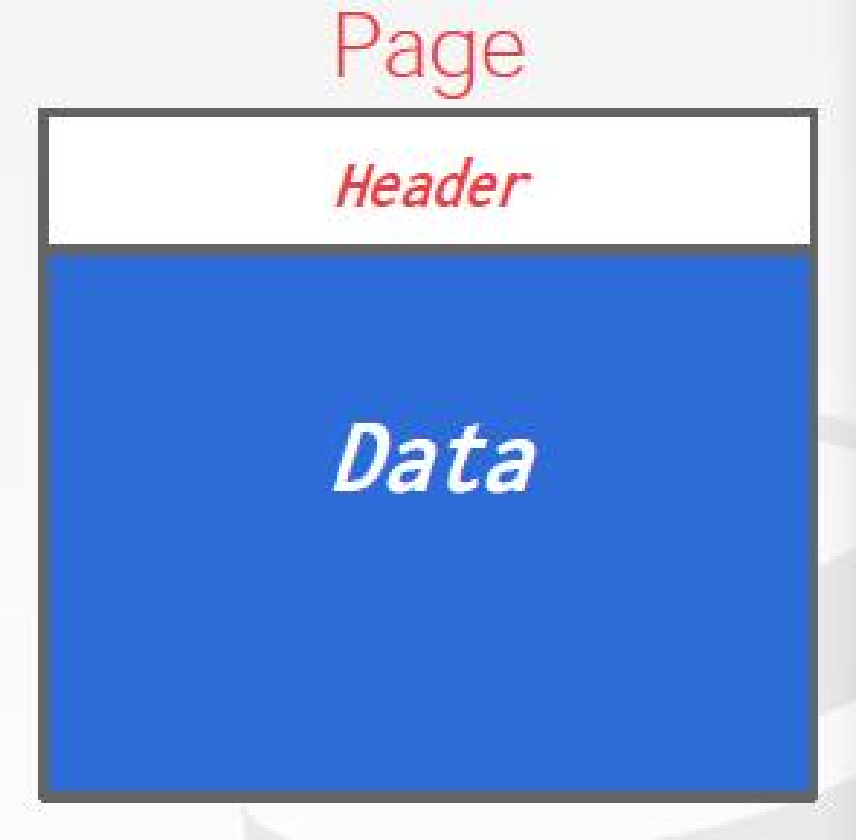
<page_id,relative_offset,free_space_size>

8.1.2.3 页的组织结构（Page Layout）

一个页面的内部结构，包括：

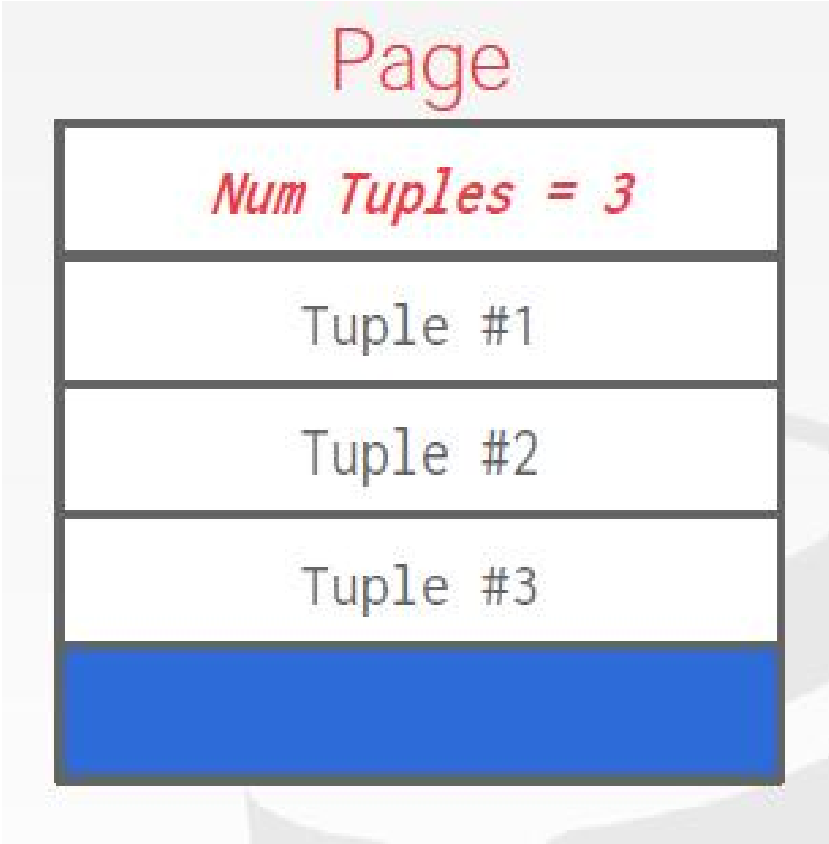
- 页头（page header），包含有关页内容的元数据信息：
  - 页大小
  - 校验和
  - DBMS版本
  - 事务可见性
  - 压缩信息

  有些系统要求页面是自包含的（如Oracle）。
- 数据区：存放数据的区域，其组织方式：
  - 面向元组型
  - 日志结构型



元组存储（Tuple Oriented）

- Strawman Idea：
  - Header：记录页内的元组数，类似数组的方式进行存储；
  - Tuple：每次添加的元组放在已有元组的后面。
- 存在的问题：
  - 删除元组时会产生碎片
  - 变长元组可能产生其他更多问题，比如元组的查询开销。
- 一般用的较少，更常见的是slotted pages（槽页）方式



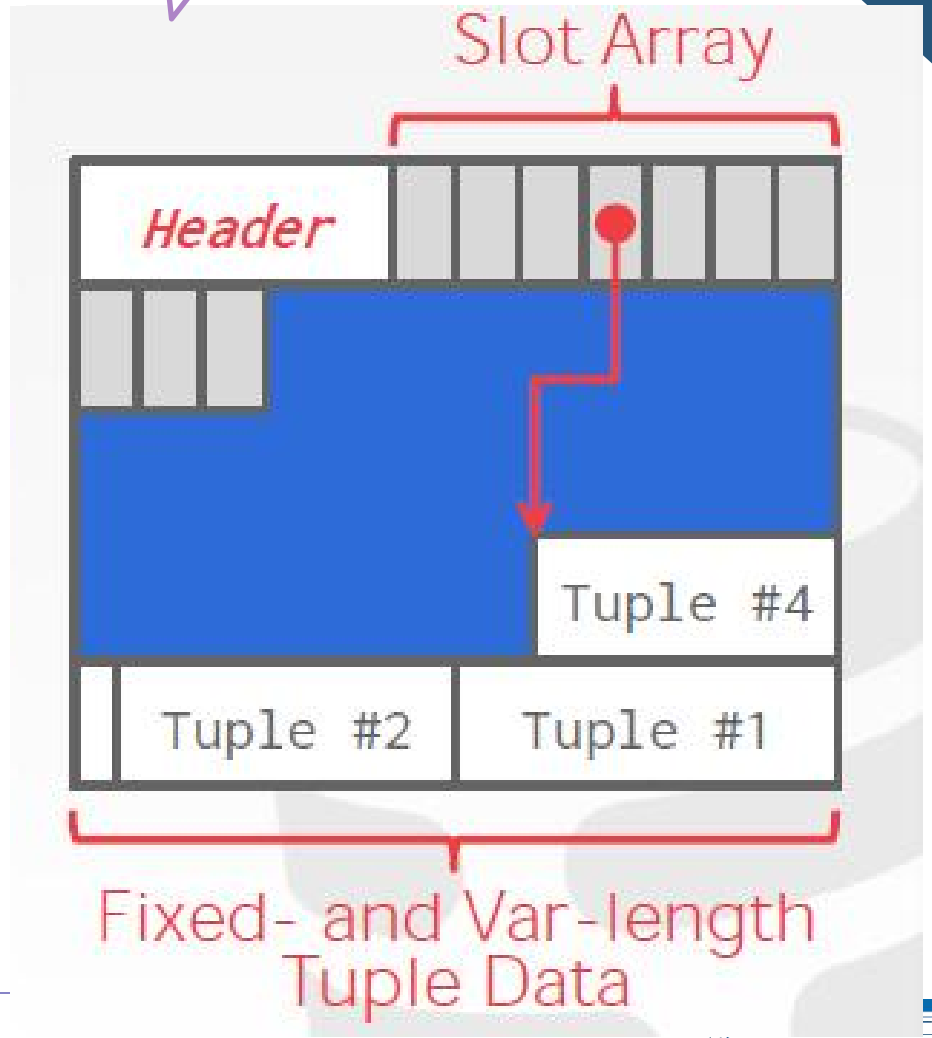
槽页（Slotted Pages）

首尾“相遇”？
页“满了”

删除？Slot数组值置-1/slot位图/压缩vacuum/写入时压缩/空置不管

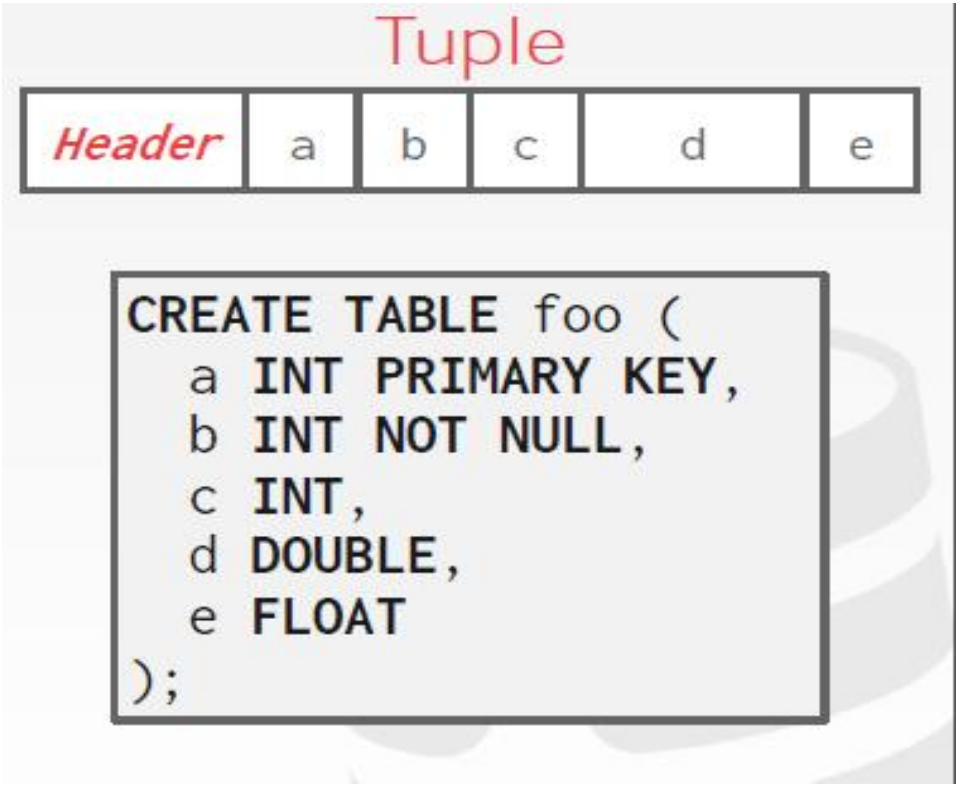
- 槽页：Slot数组将“槽位”映射到元组开始位置的偏移量。
- 当今DBMS最常用方法。
- Header记录：
  - 已占用的槽位；
  - 最后一次使用的槽的起始位置偏移；
- 元组：在页内倒序存放。
- 元组在内部的唯一标识符：

  page_id+ offset/slot，也可包含文件位置信息
- 定长、变长元组轻松应对



8.1.2.4 元组设计（Tuple Layout）

- 一个元组在页中本质上是一个“字节序列”。
- DBMS负责将这些字节解释为各个属性的类型和值。
- 记录包括：记录头，记录数据
- Tuple Header：每个元组有一个前缀为header包含元数据（例如对并发控制而言是否可见、空值的Bit Map）
  - 页中无需存放关系模式信息，专门的“catalog page”可有效减少重复信息。
- Tuple Data：实际数据，基本上按属性顺序存放。



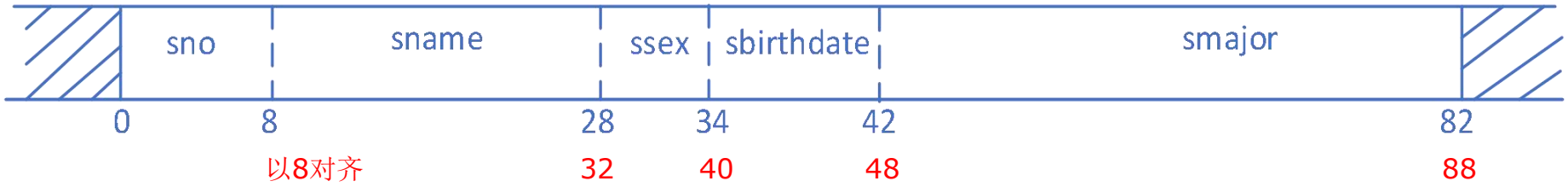
8.1.2.4 元组设计

关系表的元组可以以定长和变长两种方式存储。

- 定长元组（记录）存储

  关系中每条记录占据相同大小的空间；若有变长，保留最大空间。

- 例：Student（sno, sname, ssex, sbirthdate, smajor）



优点：快速定位，增删改比较方便、便捷。

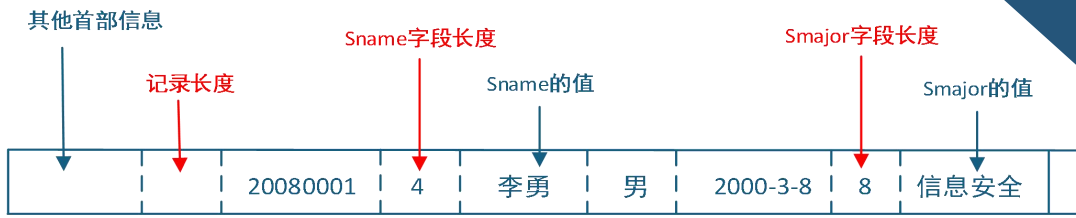
缺点：可能浪费存储空间。

有一些匹配硬件的优化动作（内存对齐问题）：字段的起始地址对齐4或者8的整数倍，以适应一些硬件系统的处理过程。

26

元组设计

- 变长记录：允许记录中存在一个或多个变长字段。技术需求：在组织变长元组时，应能保证快速访问到第一条元组，以及能快速访问其中所有字段（定长和变长字段）。

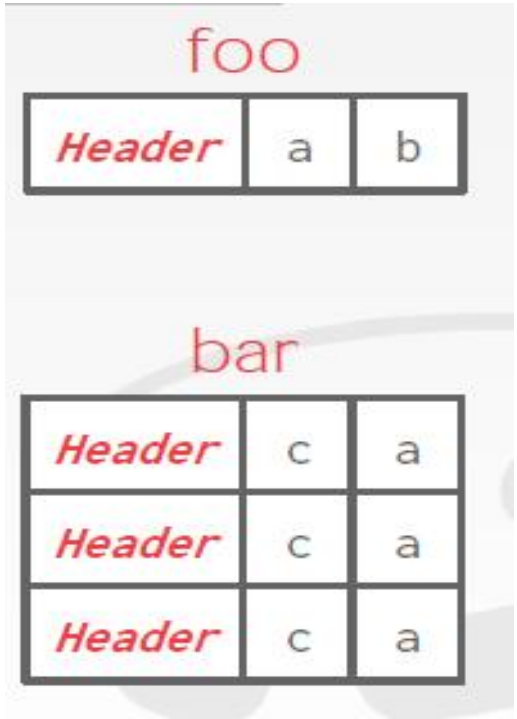


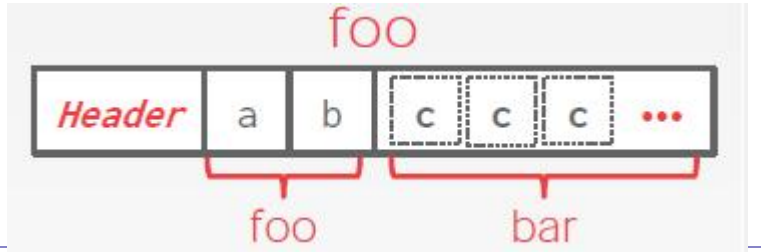
- 3种实现方式：
  - 元组头部记录元组长度，每个变长属性前记录该字段长度。
  - 将所有定长字段放在变长字段之前，记录头增加：记录长度+非1st变长字段offset
  - 保持记录定长，变长部分放在另一个溢出页

元组设计

- 物理上非规范化（Denormalize）元组设计：（“预连接”）将“相关”的元组存放在一个页或相邻页中。
  - 可以有效减少相应查询的I/O次数；
  - 也可能带来额外的数据维护开销。

```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
CREATE TABLE bar (
  c INT PRIMARY KEY,
  a INT
  REFERENCES foo (a),
);
```





8.1.2.5 关系表的组织

1 堆存储

表中的一条记录可以存储在该表的任何块中，没有顺序要求。插入时，只需在块中找到合适的空闲空间即可，若没有则申请新块。

2 顺序存储

一个表中的各条记录按照指定的属性或属性组的取值大小顺序存储。在一个块内，记录按照排序属性(组)的取值物理排列，同一个表的不同块之间则通过指针链接实现有序。

- 优点：可以高效地处理按排序属性(组)进行查询的请求。
- 例：学生表按专业(Smajor)升序存储，则可以快速完成如下的SQL语句。

  SELECT Sno, Sname, Ssex FROM Student WHERE Smajor='计算机科学与技术';
- 缺点：在插入或修改记录时为保持记录顺序，可能要在块内或块间迁移记录，导致代价较高.

29

8.1.2.5 关系表的组织

3 多表聚簇存储

- 不同表的记录聚簇存储在同一组块中，其目的是减少连接操作的开销（预连接）。（默认情况下，一个块只存储一个表中的记录，不同表的记录分别存在不同的块中）
- 例：将学生表Student和选课表SC均按照学号多表聚簇存储，从而优化以下连接查询。

```
SELECT SC. Sno, Sname, COUNT(*)    FROM Student, SC
        WHERE Student.Sno=SC.Sno AND SC. Sno='20180001';
```

- 缺点：可能导致非聚簇属性条件的查询效率变低。
- 例：

```
SELECT *  FROM Student
        WHERE Smajor='数据科学与大数据技术' AND Ssex='女';
```


30

8.1.2.5 关系表的组织

4 B+树存储

- 以B+树索引的形式确定记录存入的数据块，在保持较高的记录访问效率的同时降低数据维护开销。
- B+树存储与B+树索引的区别：叶节点数据块存放的是数据记录而不是索引项。

5 哈希存储

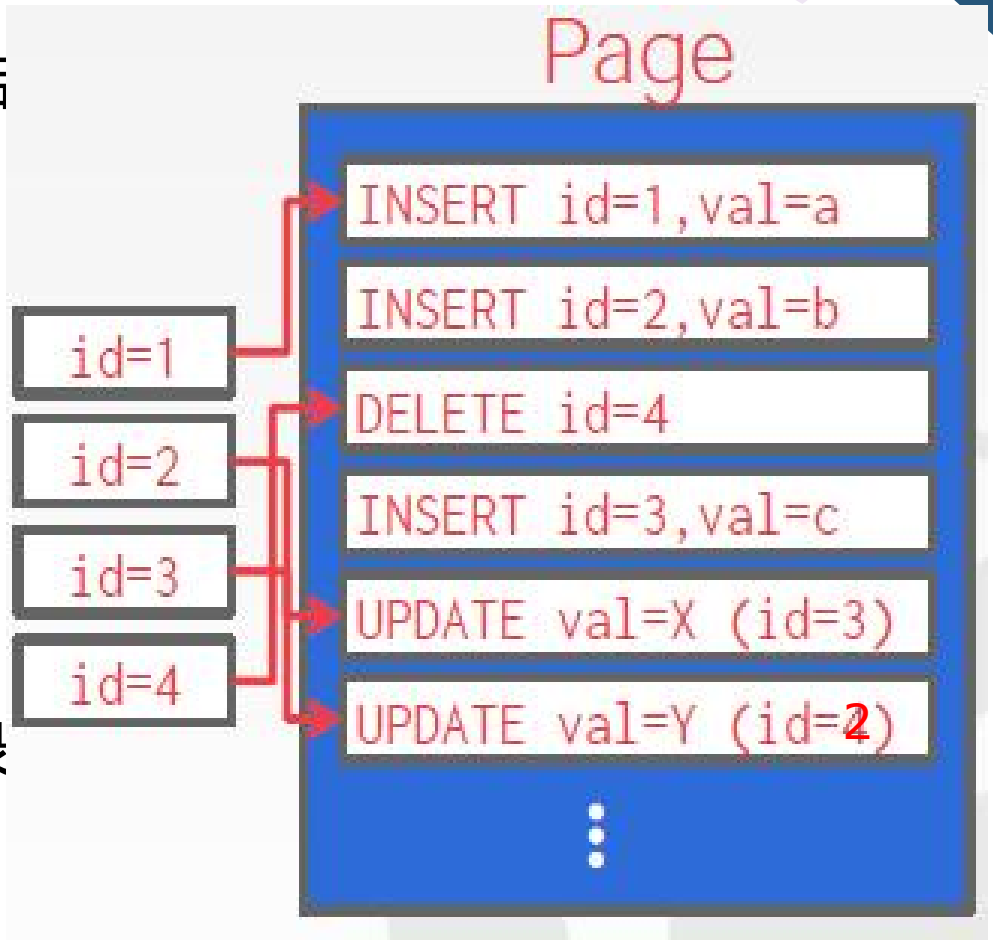
计算表中指定属性的哈希值，以此确定记录存储在哪个块。哈希存储两个关键要素：哈希表、哈希函数。（哈希属性值相同的记录会存储在相同的数据块中）

- 哈希表：由B个哈希桶组成，每个桶对应一个或多个物理块，并用0至B-1之间的一个整数作为桶编号，桶中可存储一条或多条记录。
- 哈希函数：将记录依据其哈希属性值对应到一个存储位置，函数的输入是记录的哈希属性，输出一个0至B-1之间的整数对应桶号。在查找记录时，通过计算其哈希属性值的输出快速定位到记录的位置。

31

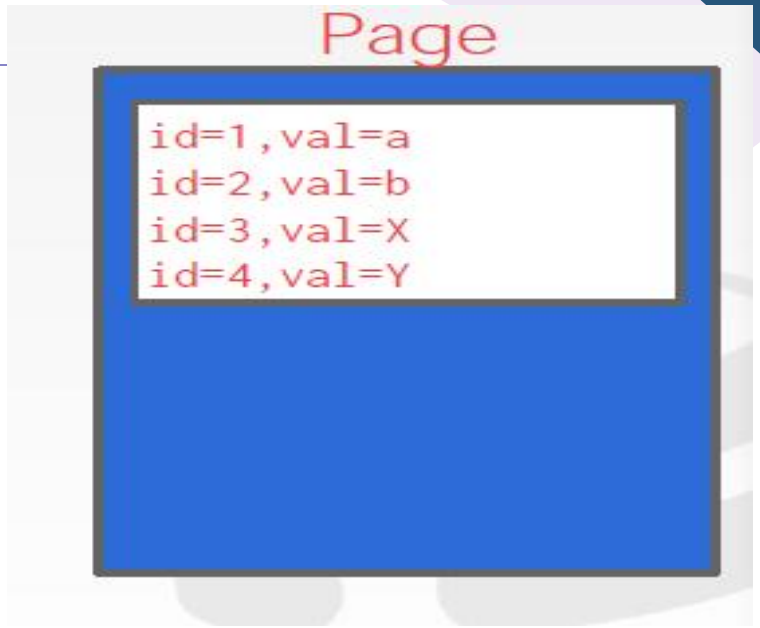
8.1.2.6 日志式(Log-Structured)文件组织

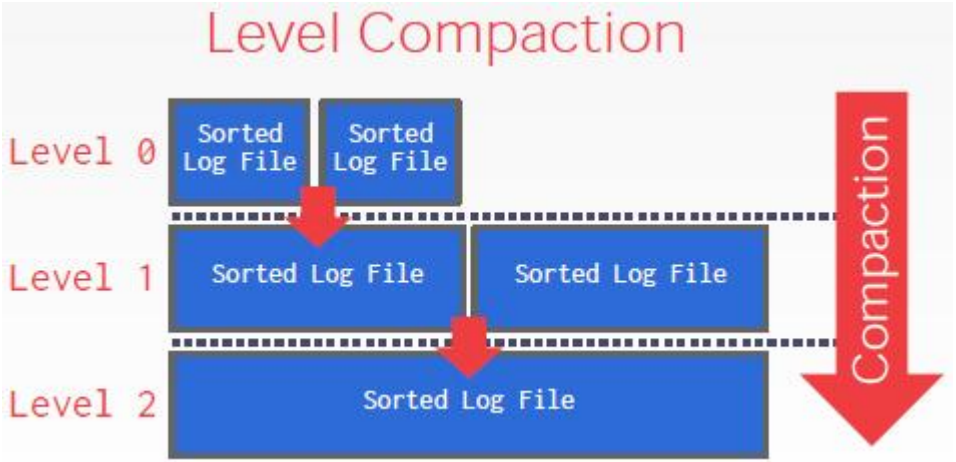
- DBMS不存储元组，而**只存储日志记录**
- 系统添加日志记录来反映数据库更新的结果
  - “插入”：存放整个元组；
  - “删除”：标记该元组被删除；
  - “更新”：记录被修改的属性的变化。
- 当需要读取日志记录，DBMS可以反向扫描日志，重新创建元组，还可“回滚”。
- 日志可定期压缩（通过删除不必要的记录来合并日志文件）。



日志文件组织

- 策略1：不同数据通常可能在不同page，如果将更新信息写到数据页，则需要访问多个page（易读优化）。
- 策略2：将数据更新信息写到一个或者连续页效率更高（读可能不连续）。
- 可建立“日志索引”，方便查找相关的日志记录。
- 定期压缩日志，去除不必要的记录。





8.1.3 索引结构

1. 顺序表索引：（1）稠密索引（2）稀疏索引（3）多级索引
2. 辅助索引
3. B+树索引（增加：CMU树索引）
4. 哈希索引（增加：CMU散列表）
5. 位图索引

数据库管理系统内部的数据结构

- 内部元数据
- 核心数据存储
- 临时数据结构
- 表索引：是表属性子集的副本，是对这些属性的组织及排序，目的是为了使用这些属性的子集进行高效的访问（主要针对非顺序扫描方式）。
  - 索引的优势

    ① 表索引块数量通常比数据块数量少得多，因而搜索起来比较快。

    ②)索引数据结构，有利于使用高效的方法在索引中快速查找。

    ③对于经常访问的表，若其索引文件足够小则可让其长驻内存，减少I/O。
  - 索引带来的额外开销：存储空间、建立、维护索引的开销。

35

数据库管理系统内部的数据结构

- DBMS关于索引的任务：
  - 确保表的内容和索引在逻辑上是同步的；
  - 执行每个查询时选出最佳的索引；
  - 在索引的数量和开销上进行权衡：存储开销、维护开销

- 数据库中常见的索引结构包括：
  - 顺序表索引
  - 辅助索引
  - B+树索引
  - 哈希索引
  - 位图索引等。

36

8.1.3 索引结构

8.1.3.1 顺序表索引

- 为了提高顺序表的查找效率，针对关系表的顺序存储方式，可以在顺序表的排序属性(组)上建立索引。
- 由于这种索引结构是建在按索引属性值顺序存储的关系表上的，也称作主索引(primary index)，也是聚簇索引(clustering index)的一种形式。

  注意:虽然称作主索引，但这种索引结构与主码没有关系，索引属性可以是任意属性(组)。索引结构中，每个索引项由“索引属性的取值、指向相应记录的指针（或者记录本身）”两部分组成。
- 顺序表上的主索引分为稠密索引和稀疏索引两类。
- 主索引相对应的是辅助索引(secondary index) 。

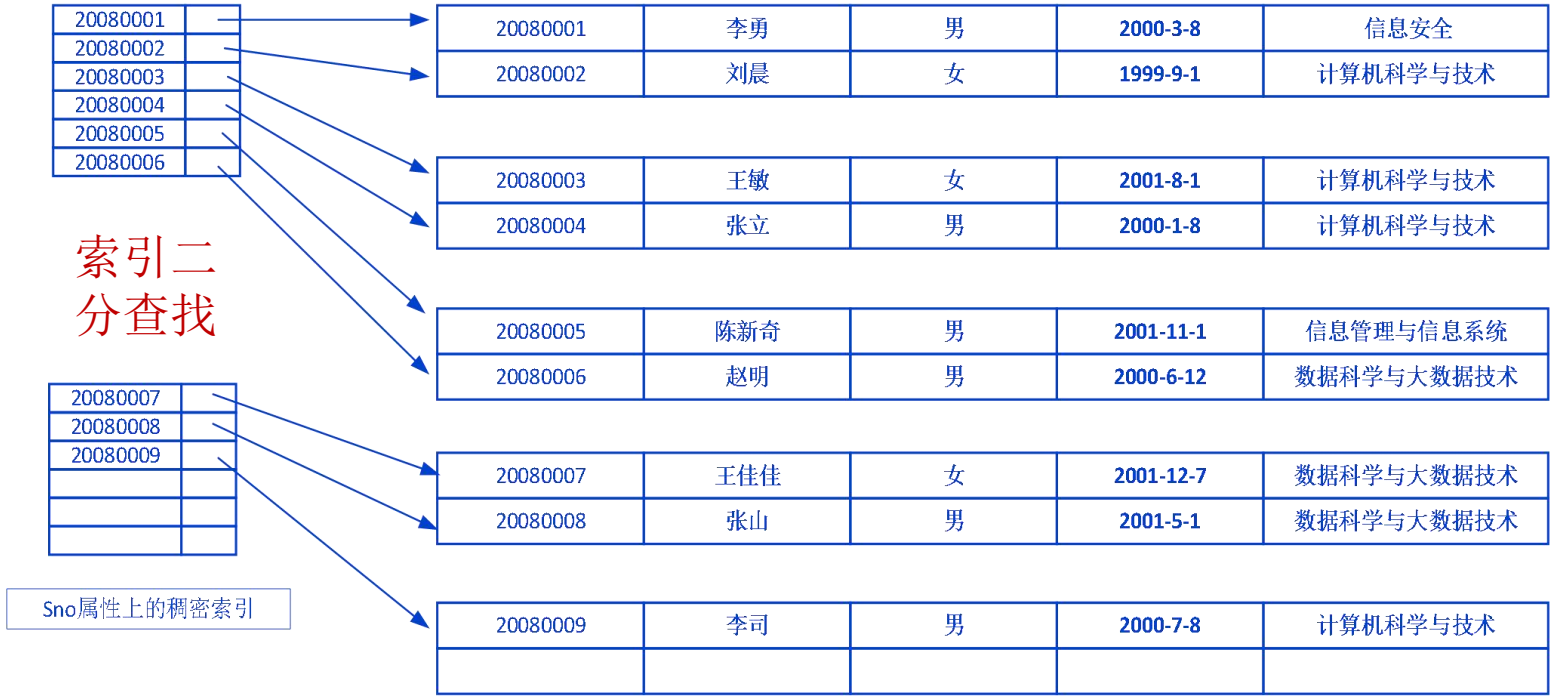
37

8.1.3.1.1 稠密索引

- 稠密索引(dense index)的索引块中存放每条记录的索引属性值以及指向相应记录的指针。
- 例：Student表Sno属性上的稠密索引，设每个块只能存放2条Student表记录或6个Sno索引项，如图所示。其中，每条Student记录在索引中都对应了一个索引项。
  - 查找某学生信息：

```
Select *
From Student
WHERE Sno='20180012';
```

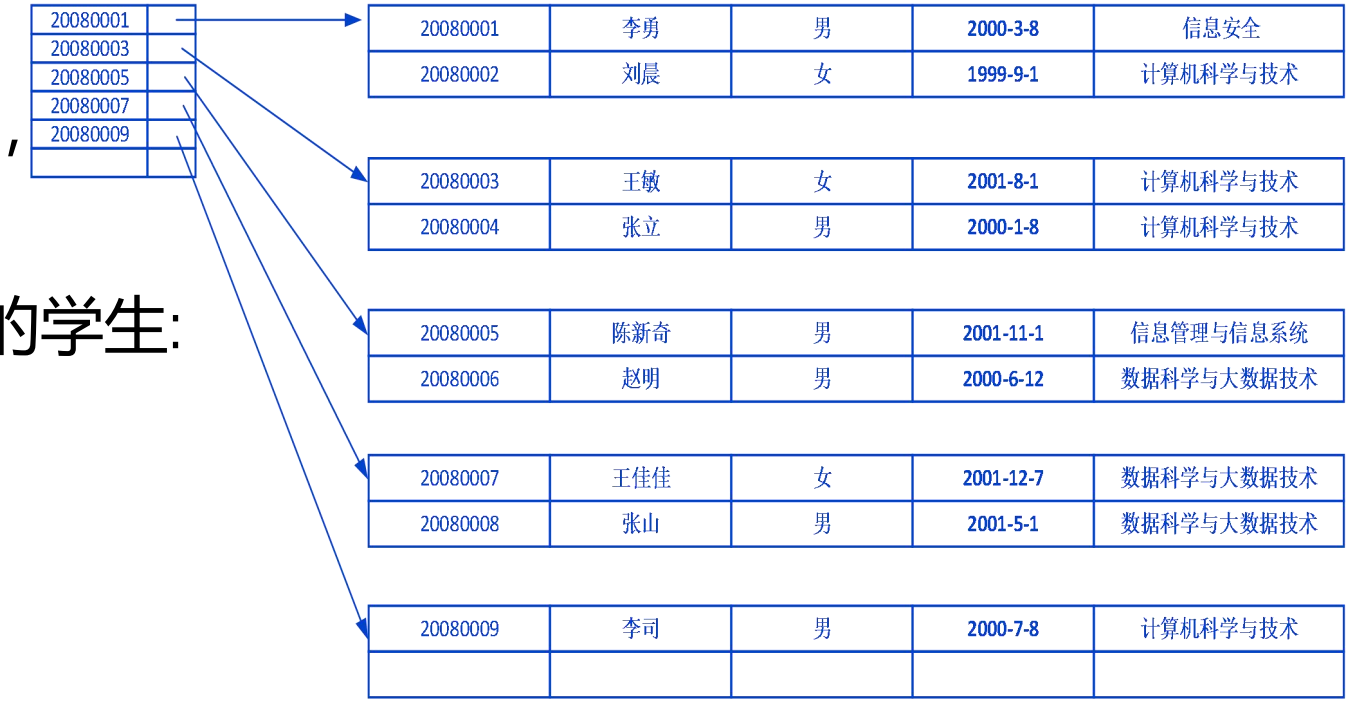
借助稠密索引：3次IO；
不借助：6次IO



38

8.1.3.1.2 稀疏索引

- 问题：稠密索引在关系的记录数较多导致索引比较大，从而对索引查找会花费较多时间，引入稀疏索引。
- 稀疏索引(sparse index) 中，基本表的每个物理存储块只对应一个索引项，即每个索引项存放每个物理块的第一条记录的索引属性值及指向该物理块的指针。
- 例：Student表稀疏索引如图所示，图中索引的块数少于稠密索引。



- 查询Student表中学号为20180012的学生:

  借助稀疏索引：2次IO；

- 另一大优势：索引维护代价低。

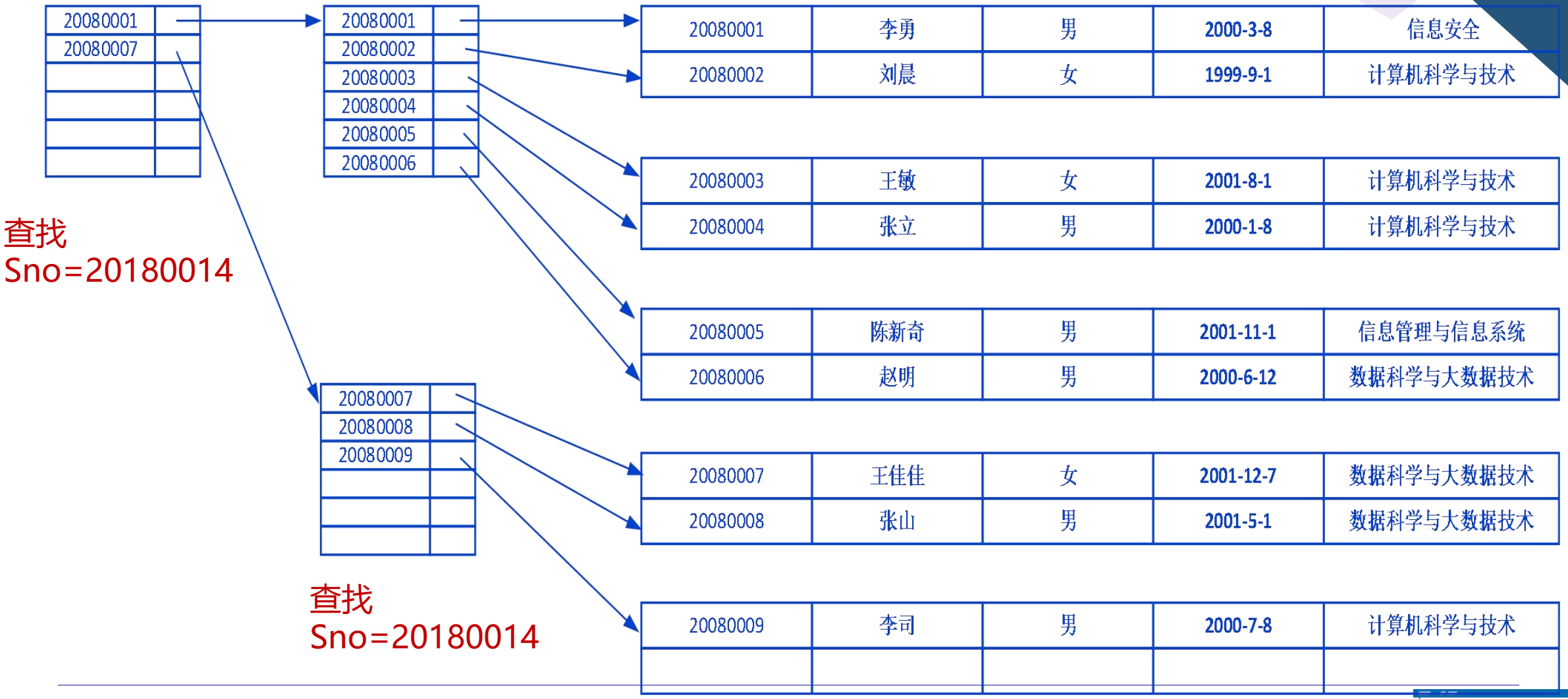
8.1.3.1.3 多级索引

- 问题：当关系表的数据量非常大时，稀疏索引也可能比较大，导致读取稀疏索引及在其中查找的效率降低，针对此引入多级索引。
- 多级索引(multilevel index)中，
  - 第一级索引（最底层）是稠密索引或稀疏索引；
  - 当这级索引较大时，可以在其上再建第二级索引。
  - 如果第二级索引仍然较大，可以在第二层索引上建第三级索引。
  - 以此类推，直到索引尺寸合适为止。
  - 二级索引或更高级的索引必须是稀疏索引。

- 例：Student表上建立的二级索引如下页图所示，其中二级为稀疏索引。

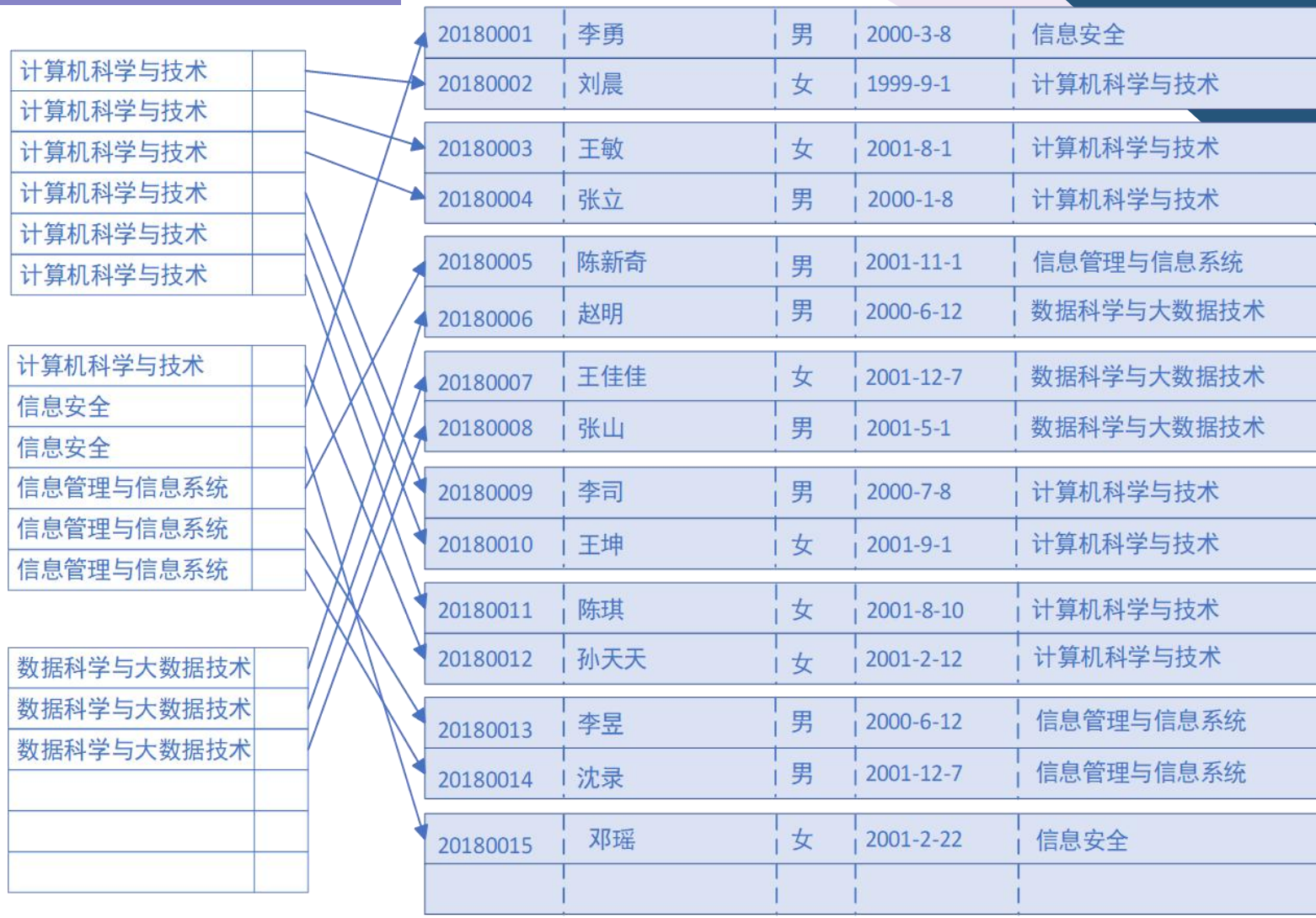
40

例如：Student表上建立的二级索引



8.1.3.2 辅助索引

- 辅助索引(secondary index)是建立在表的非排序属性上的索引。由于辅助索引是建在无序属性上的，所以它必须是稠密索引。
- 一个表最多只能建立一个主索引，但可在不同的属性上建立多个辅助索引。
- 例：Student表的Smajor属性上建立的辅助索引如图所示。



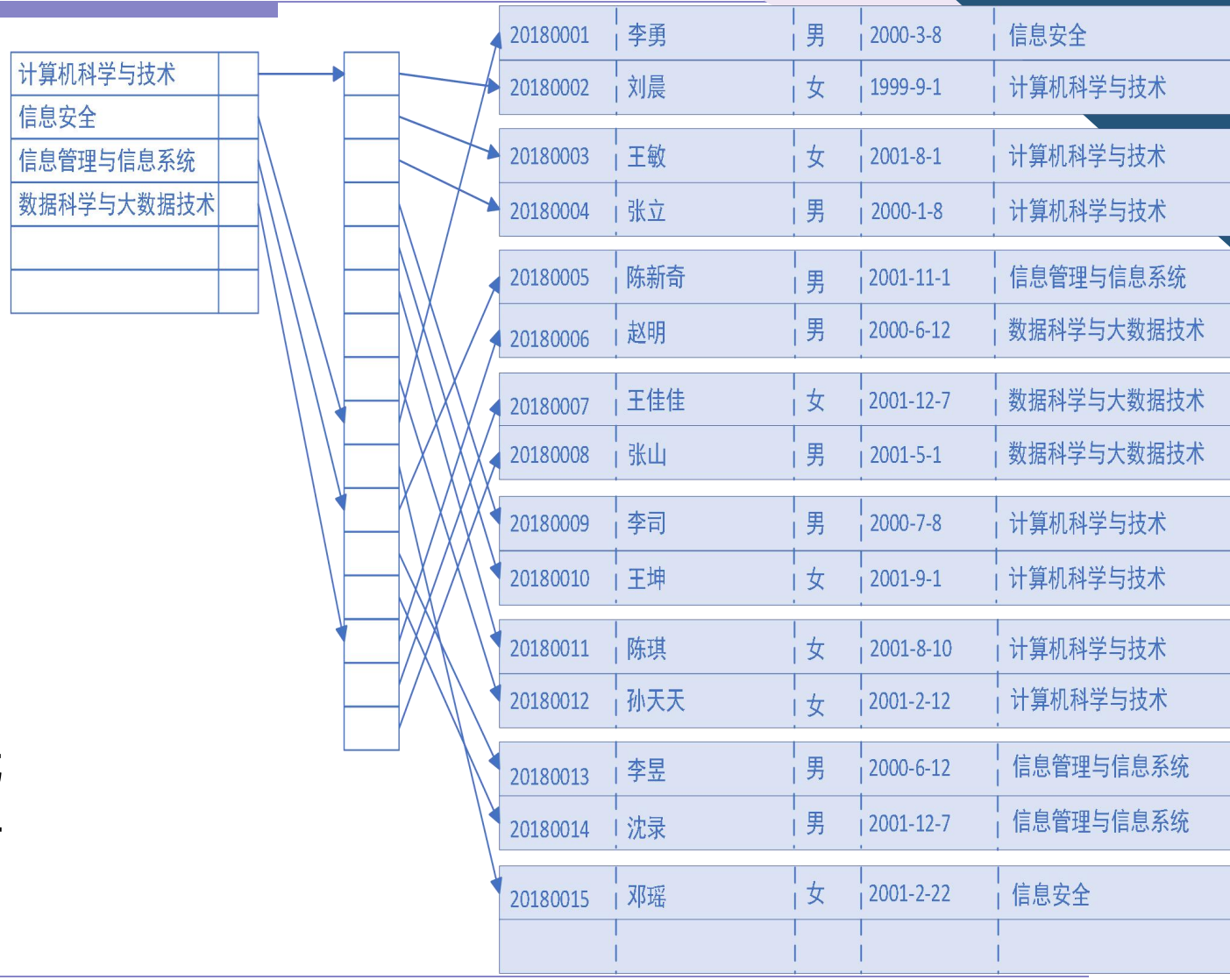
42

8.1.3.2 辅助索引

- 辅助索引的问题——重复值

建有辅助索引的属性往往会取重复值。去掉重复取值的索引项，可以减小索引大小，从而减少索引查找的开销。但是，不同索引属性值重复次数往往不相同，很难对其预先估计，因此不能简单通过在每个索引项中预留多个指针来解决这个问题。

- 指针桶：桶中的指针指向相应元组，而索引项中的指针指向指针桶中的相应位置，如图所示。



43

8.1.3.2 辅助索引

- 指针桶对于关系表上多个辅助索引的优化

  ——可以利用辅助索引的指针桶来回答涉及多个属性条件的查询。

- 例：对于如下查询

```
SELECT *
FROM Student
WHERE Smajor='计算机科学与技术
AND Sbirthdate='2002-1-1';
```

如果Student表在Smajor属性和Sbirthdate属性上都建有辅助索引，则可以分别利用两个辅助索引找出满足各自条件的指针组，对两组指针求交集即可得到同时满足两个条件的指针组，这组指针所指向的元组就是查询结果。

44