

# 数据库系统原理 (CS307)

## 第十二讲:存储系统与结构

Zhong-Qiu Wang

计算机科学与工程系  
南方科技大学

· 大部分内容来自 Stéphane Faroult 和《数据库系统概念（第 7 版）》作者制作的幻灯片。 · 他们的原始幻灯片已修改以适应南方科技大学 CS307 的时间表。 · 幻灯片主要基于马宇欣博士提供的幻灯片

# 物理存储系统

# 物理存储系统

·记录数据的硬件



# 物理存储介质的分类

- 可以将存储区分为：

- **易失性存储** · 断电时

- 内容丢失 · **非易失性存储**：

- 即使电源关闭,内容仍会保留 · 包括二级和三级存储,以及电
      - 池备份的主存储
      - 记忆

- 影响存储介质选择的因素包括 · 数据访问速度 · 每单位数据的成本 · 可  
靠性

# 存储层次结构

## ·主存储

- 速度最快的介质但易失性（缓存、主存储器）。

## ·二级存储

- 层次结构的下一级,非易失性,访问时间适中
  - ... 也称为在线存储 例如闪存、

磁盘

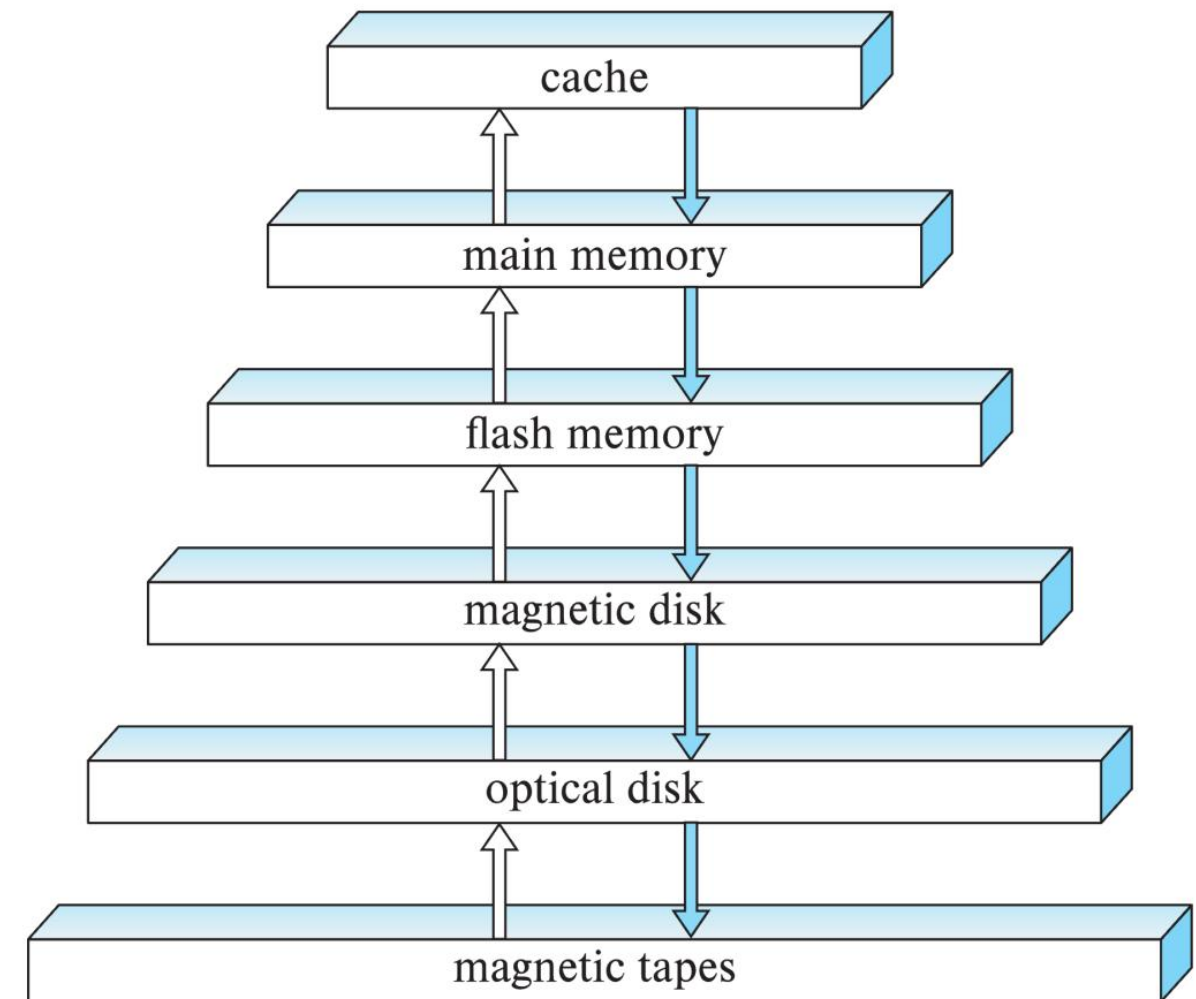
## ·三级存储

- 层次结构中的最低级别、非易失性、访问时间较慢
  - ... 也称为离线存储,用于档案存储

- 例如磁带、光存储（DVD、蓝光 DVD）

### ·磁带

- 顺序访问,容量为 1 至 12 TB
- 几个驱动器,带很多磁带
- 具有 PB 级（数千 TB)存储空间的点唱机



# 存储接口

## · 磁盘接口标准系列

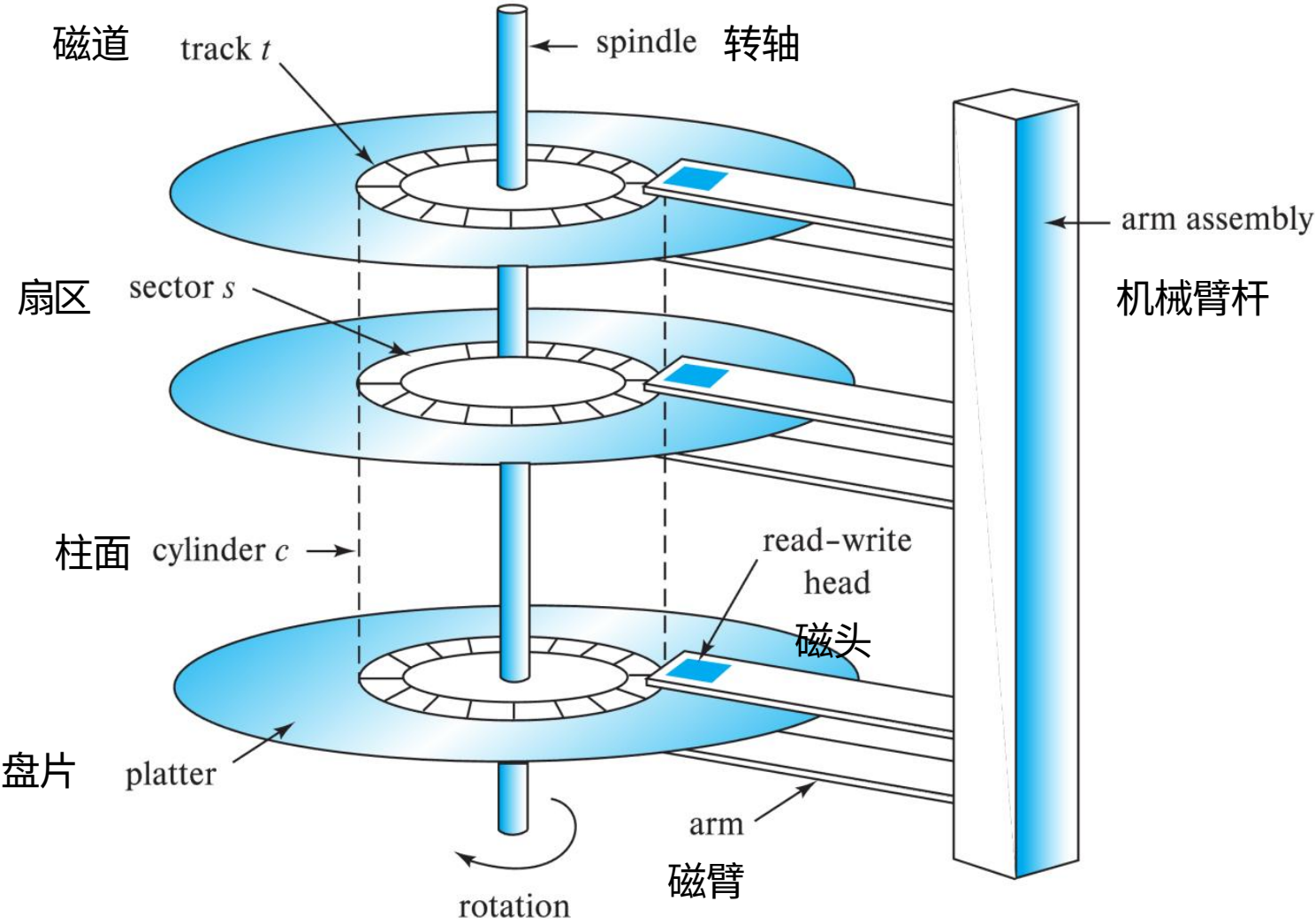
- SATA (串行 ATA)
  - SATA 3 支持高达 6 千兆位/秒的数据传输速度
- SAS (串行连接 SCSI)
  - SAS 版本 3 支持 12 千兆位/秒
- NVMe (非易失性存储器快速)接口
  - 与 PCIe 连接器配合使用,支持更低的延迟和更高的传输速率
  - 支持高达 24 千兆位/秒的数据传输速率

## · 磁盘通常直接连接到计算机系统,但是……

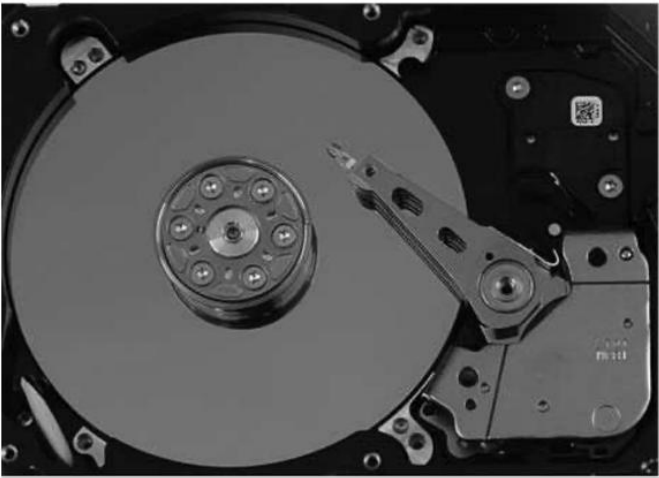
- 在存储区域网络 (SAN) 中,大量磁盘通过高速网络连接到服务器数量
- 在网络附加存储 (NAS) 中,网络存储使用网络提供文件系统接口  
文件系统协议,而不是提供磁盘系统接口



# 磁性硬盘机制



磁盘驱动器示意图



磁盘驱动器的照片

# 磁性硬盘机制

## ·读写头

## ·盘片表面被分成多个环形轨道

·典型硬盘的每个盘片上有超过 50K-100K 个磁道 ·每个磁道被划分为扇区

扇区是可以读取或写入数据的最小单位

·扇区大小通常为 512 字节（现代操作系统需要 4KB） ·每条磁道的典型扇区数：

500 到 1000 个（内磁道）到 1000 到 2000 个（外磁道）

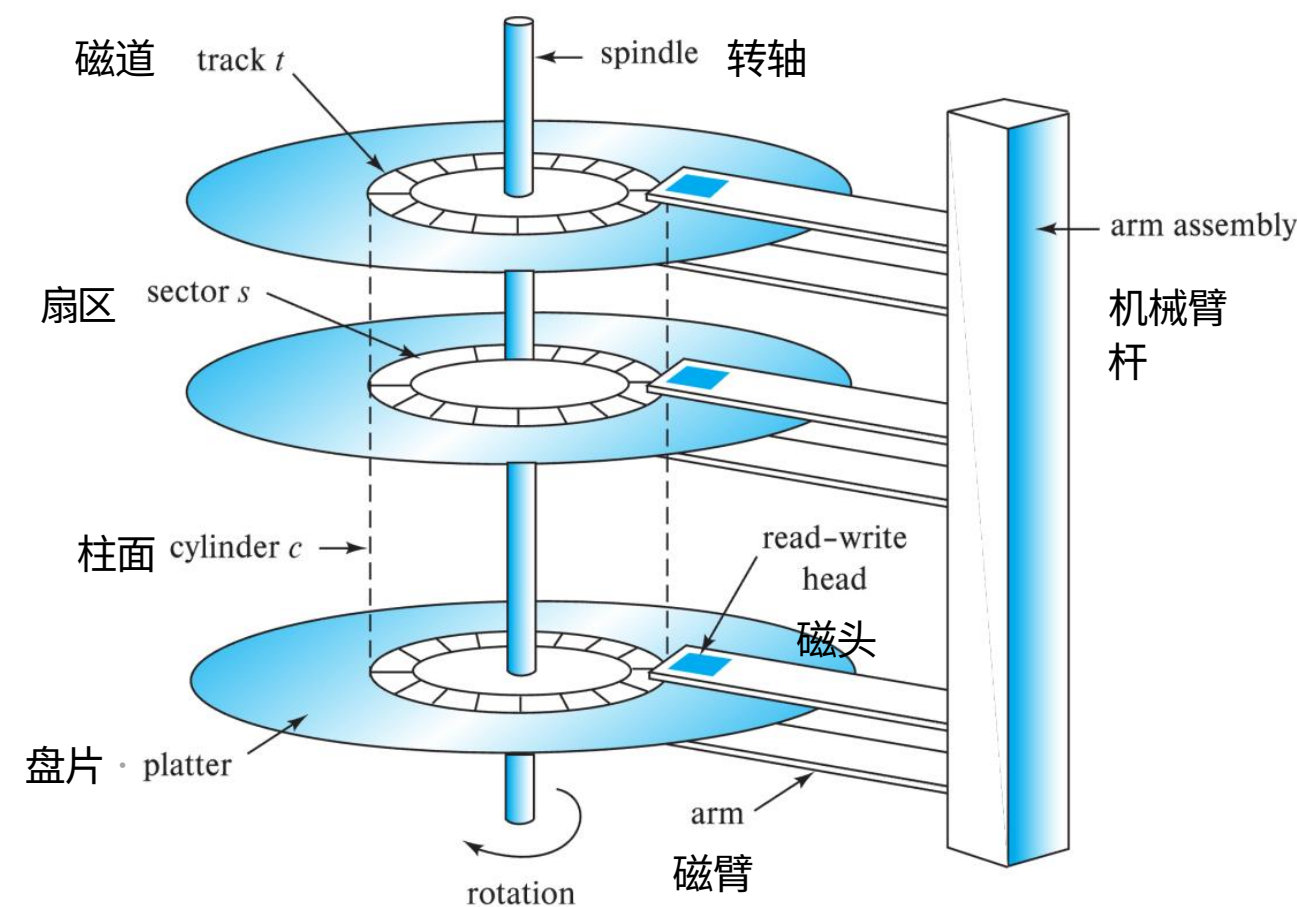
## ·读取/写入扇区

·磁盘臂摆动以将磁头定位在正确的轨道上 ·盘片不断旋转;当扇区通

过磁头下方时,数据被读取/写入 ·磁头磁盘组件

·单个主轴上有多个磁盘盘片（通常为 1 到 5 个） ·每个盘片一个磁头,安装在一个公共臂上 ·第  $i$  个柱面由所有盘片的第  $i$  个

磁道组成





# 磁性硬盘机制

·**磁盘控制器**: 计算机系统与磁盘之间的接口  
驱动硬件

· 接受高级命令来读取或写入扇区 · 发起操作, 例如将磁盘臂移动到正确的轨道并读取或写入数据

· 在写入过程中计算并附加校验和 (校验和) 到每个扇区

· 它可以验证稍后读回的数据是否正确

· 如果数据损坏, 则存储的校验和很可能与重新计算的校验和不匹配

· 写入扇区后, 通过读回扇区来确保写入成功

· 执行坏扇区重新映射

· 将坏扇区的地址映射到好扇区的地址

# 磁盘的性能指标

·**访问时间**:从发出读取或写入请求到执行所花费的时间

数据传输开始时。包括: ·寻道时间 – 将机械臂重

新定位到正确轨道上所需的时间

- 平均寻道时间是最坏情况寻道时间的 1/2

- 如果所有磁道都有相同数量的扇区,并且我们忽略启动和停止的时间,则为 1/3  
手臂运动

- 典型磁盘上为 4 到 10 毫秒

·**旋转延迟** – 被访问的扇区出现在磁头下方所需的时间

- 典型磁盘 (5400 至 15000 rpm)上为 4 至 11 毫秒

- 平均延迟是上述延迟的 1/2

·总体延迟为 5 到 20 毫秒,具体取决于磁盘型号

·**数据传输速率**:从设备检索数据或将数据存入设备的速率

磁盘

- 最大速率为每秒 25 到 200 MB,内轨 (扇区较少)的速率较低

# 磁盘的性能指标

·每个请求指定要引用的磁盘地址,该地址采用**块号**的形式

·**磁盘块**是存储分配和检索的逻辑单元

·通常为 4 至 16 千字节

·更小的块:更多来自磁盘的传输

·更大的块:由于块部分填充而浪费更多空间

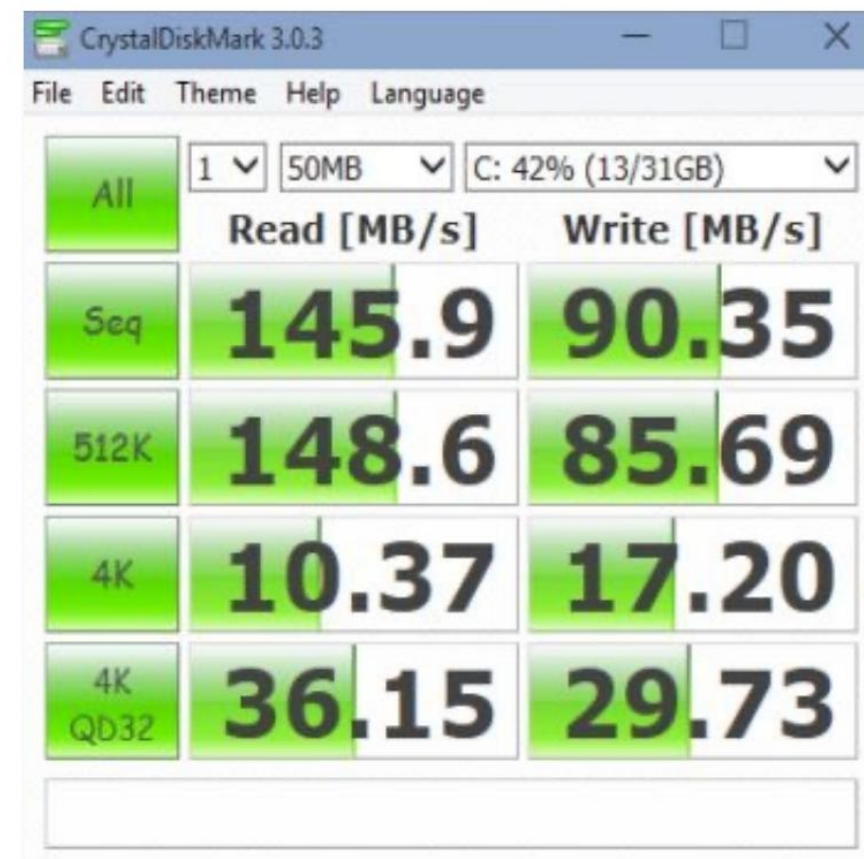
·**顺序访问模式** ·连续请求针对连续

的磁盘块 ·仅需要第一个块的磁盘寻道 ·**随机访问模式**

·连续请求针对的是磁盘上任意位置的块 ·每次访问都需要寻道 ·传输速率较低,因为大量时间浪费在寻道上

·每秒 I/O 操作数(IOPS)

·磁盘每秒可支持的随机块读取次数 ·当前一代磁盘上为 50 到 200 IOPS



# 磁盘的性能指标

- **平均故障时间 (MTTF)** – 磁盘预计连续运行而不发生任何故障的平均时间（即可靠性）
  - 通常为 3 至 5 年
  - 新磁盘发生故障的概率相当低,相当于“理论新磁盘的“MTTF”为 500,000 至 1,200,000 小时
    - 例如,新磁盘的 MTTF 为 1,200,000 小时,这意味着给定 1000 个相对新磁盘,平均每 1200 小时就会出现一次故障
  - MTTF 随着磁盘老化而减少

# 闪存

- NOR 闪存与 NAND 闪存

- NAND 闪存

- 广泛用于存储,比 NOR 闪存便宜 · 需要一次读取一页 (页面:512 字节

- 到 4 KB) · 读取一页需要 20 到 100 微秒 · 顺序读取和随机读取之间没有太大

- 区别 · 页面只能写入一次

- 必须擦除才能重写

- 固态硬盘 (SSD)

- 使用标准的面向块的磁盘接口,但将数据存储在内部的多个闪存设备上

- 使用 SATA 时传输速率高达 500 MB/秒,使用 NVMe PCIe 时传输速率高达3 GB/秒

# 闪存

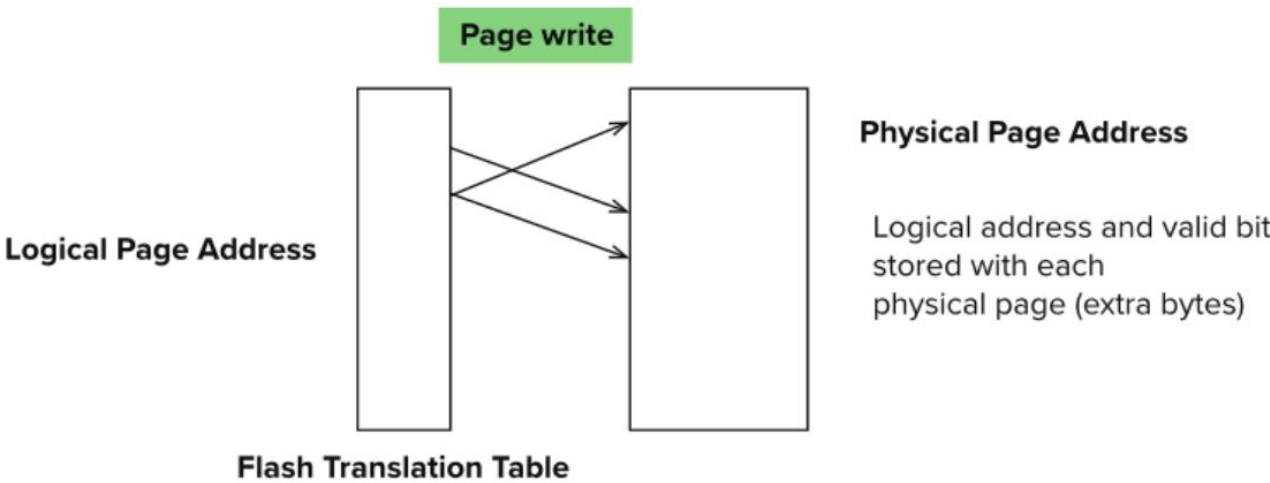
- 擦除以擦除块为单位进行

- 需要 2 到 5 毫秒

- 擦除块通常为 256 KB 到 1 MB（128 到 256 页）

- 将逻辑页面地址重新映射到物理页面地址,避免等待擦除 ·闪存转换表跟踪映射 ·还存储在闪存页面的标签字段中 ·重新

映射由闪存转换层执行



- 经过 100,000 到 1,000,000 次擦除后,擦除块变得不可靠且无法使用

- 磨损均衡（平衡每个块被擦除的次数） ·可以使用重新映射将数据存储在好块中



# SSD 性能指标

- 每秒随机读取/写入次数

- 典型的 4KB 读取 :每秒 10,000 次读取 (10,000 IOPS)
  - 典型的 4KB 写入 :40,000

- IOPS
  - SSD 支持并行读取
  - 典型的 4KB 读取 :

- SATA 上 100,000 IOPS,32 个并行请求 (QD-32)

- NVMe PCIe 上 QD-32 性能达 350,000 IOPS

- 典型的 4KB 写入 :

- QD-32 可提供 100,000 IOPS,某些型号甚至更高

- 顺序读/写的数据传输率

- SATA3 为 400 MB/秒,使用 NVMe PCIe 为 2 至 3 GB/秒

- 混合磁盘 :将少量闪存缓存与较大磁盘相结合

# 磁带

- 保存大量数据并提供高传输速率

- DAT (数字音频磁带)格式占用几 GB,DLT (数字线性磁带)格式占用 10-40 GB,Ultrium 格式占用 100 GB 以上,Ampex 螺旋扫描格式占用 330 GB

- 传输速率从几 MB/秒到几十 MB/秒

- 磁带很便宜,但驱动器的成本很高 ·与磁盘和光盘相比,

## 访问时间非常慢

- 仅限于顺序访问。 · 某些格式 (Accelis)

- 以较低的容量为代价提供更快寻道速度 (10 秒)

- 主要用于备份、存储不常用的信息以及作为将信息从一个系统传输到另一个系统的离线介质。

---

- 用于大容量存储的磁带点唱机

- 数 PB (10<sup>15</sup>字节)

# 袭击

- **RAID:独立磁盘冗余阵列**

- 管理大量磁盘的磁盘组织技术,提供对 单磁盘

- 通过并行使用多个磁盘实现高容量和高速度

- 通过冗余存储数据实现高可靠性,即使某个磁盘发生故障,数据也可以恢复

生故障的概率远远高于

特定单个磁盘发生故障的可能性。

- 例如,一个有 100 个磁盘的系统,每个磁盘的 MTTF 为 100,000 小时 (约 11 年) ,将系统平均无故障时间为 1000 小时 (约 41 天)

- 使用冗余技术来避免数据丢失对于大量磁盘

# 通过冗余提高可靠性

- 冗余 存储可用于重建信息的额外信息  
磁盘故障导致丢失

- 例如,镜像 (或阴影)

- 复制每个磁盘。逻辑磁盘由两个物理磁盘组成。
- 每次写入都在两个磁盘上进行
  - 可以从任一磁盘进行读取
- 如果一对磁盘中的一个磁盘发生故障,另一个磁盘上的数据仍然可用
  - 仅当磁盘发生故障,且其镜像磁盘在系统修复前也发生故障时,才会发生数据丢失
  - 组合事件发生的概率非常小 – 除火灾、建筑物倒塌或电涌等相关故障模式外

- 平均数据丢失时间取决于平均故障时间和平均修复时间

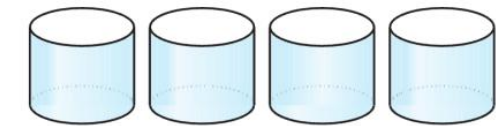
- 例如,MTTF 为 100,000 小时,平均修复时间为 10 小时,则平均数据丢失时间为  
一对镜像磁盘的使用寿命为  $500 \times 10^6$  小时 (或 57,000 年) (忽略相关故障模式)

# RAID 级别

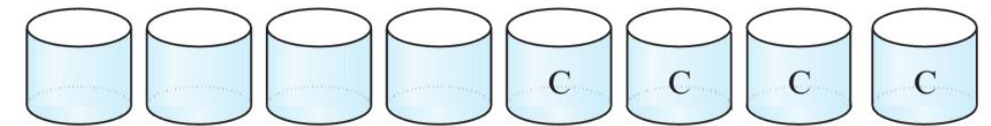
- 通过使用磁盘条带化结合以较低成本提供冗余的方案具有奇偶校验位 ·

不同的 RAID 组织或 RAID 级别具有不同的成本、性能和可靠性特征

- **RAID 0**: 块条带化 (例如, 位级条带化) ; 非冗余 · **RAID 1**: 带块条带化的镜像磁盘 · **RAID 10**: 条带化和镜像的组合

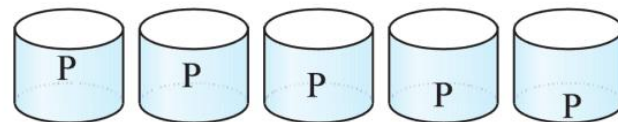


(a) RAID 0: nonredundant striping



(b) RAID 1: mirrored disks

- **RAID 5**: 块交错分布式奇偶校验 · **RAID 6**: P+Q 冗余方案



(c) RAID 5: block-interleaved distributed parity

# 磁盘块访问优化

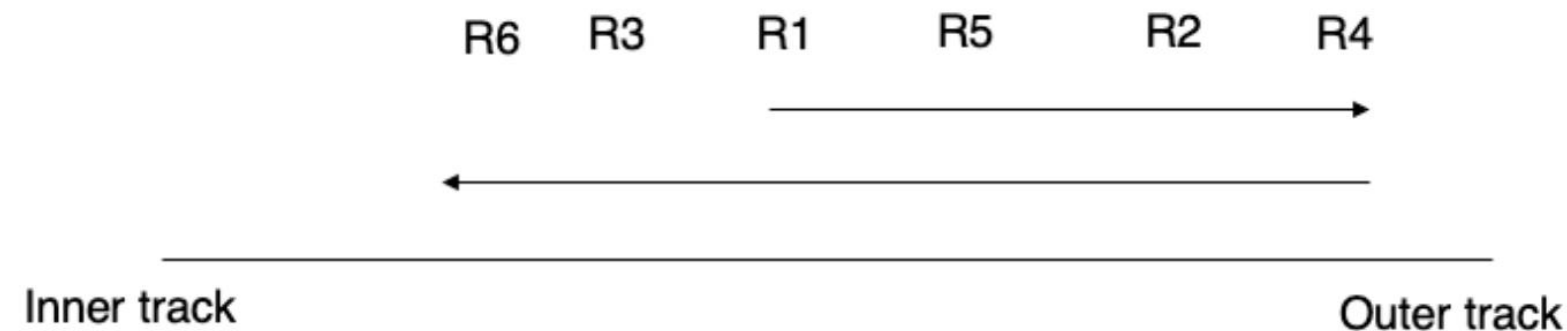
·缓冲 ·内存缓

冲区来缓存磁盘块 ·预读

·从磁道读取额外的块,以防它们很快被请求 ·适用于顺序访问 ·磁盘臂调度算法 ·重新排序块

请求,以便最小化磁盘臂移动 ·例

如,电梯算法 (像电梯一样)





# 磁盘块访问优化

## ·文件组织

- 尽可能以连续的方式分配文件块 ·以范围为单位进行分配

- 文件可能会产生碎片 ·例如,如果磁

盘上的可用块分散,并且新创建的文件的块分散在磁盘上 ·对碎片文件的顺序访问会导致磁盘臂移动增加 ·某些系统具有对文件系统进行碎片整理的实用程序,以加快文件访问速度

## ·非易失性写入缓冲区

- 暂时存储写入的数据……并立即通知操作系统写

- 入已完成且无错误 ·空闲时将数据写入磁盘

- 进行了一些优化

# (逻辑)数据存储结构

# 文件组织

- 数据库以文件集合的形式存储
  - 每个文件都是一系列记录 ·一条记录是一系列字段 ·一种方法 ·假设记录大小是固定的
  - 每个文件仅包含一种特定类型的记录 ·不同的文件用于不同的关系
- \*
  - 这种情况最容易实现 ;稍后我们将考虑可变长度记录
- 磁盘和 SSD 是块结构设备 ·数据以块 (4-8 KB)为单位读取或写入
- 数据库处理的是记录,记录通常比块小得多 ·我们假设记录比磁盘块小
- 每条记录完全包含在单个块中

# 文件组织

- 目标: **时间**和**空间** · 尽可能快地支持 CURD 操作 · 尽可能节省存储空间 · 同时在一定程度上保持数据完整性

# 固定长度记录

- 简单方法：
  - 从字节 $n \times i$
  - 1)开始存储记录 $i$  ,其中 $n$ 是每个记录的大小
  - 记录访问很简单,但如果块大小不是 $n$ 的倍数,则记录可能会跨块

bytes

修改:不允许记录跨越块边界

- 将块大小（例如 4 KB)除以记录大小（例如 53 字节）,并丢弃小数部分

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

# 固定长度记录

·删除记录*i*

·方法 #1:移动记录*i + 1, ..., n - 1*

*n*到*i, ...,*

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000



# 固定长度记录

·删除记录i

·方法 #2:将记录n移动到i 删除记录 3 并用记录 11 替换

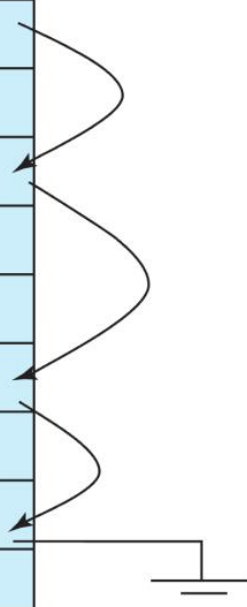
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

# 固定长度记录

·删除记录i

·方法 #3:不移动记录,但将所有空闲记录链接到一个空闲列表上

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000



# 可变长度记录

·数据库系统中可变长度记录的出现有多种方式:

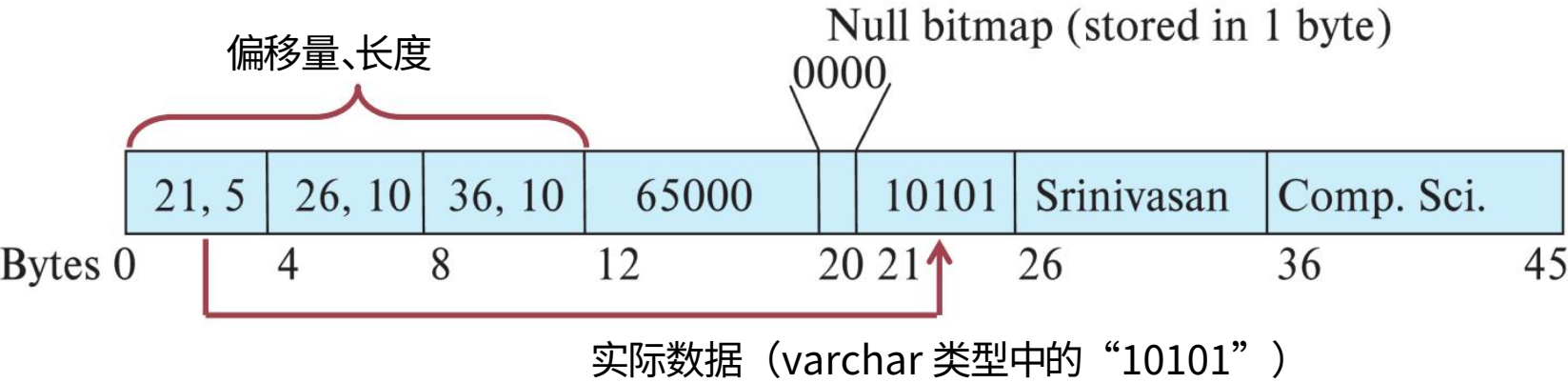
·允许一个或多个字段具有可变长度的记录类型,例如字符串(**varchar**) ·在一个文件中存储多种记录类型 ·允许重复字段的记录类型 (某些较旧的数据模型中使用的数组、多集) 。

·可变长度记录的问题 ·我们如何才能以简单的方式检索数据而不浪费太多空间 · `varchar(1000)`

– 我们真的需要为该字段分配 1000 个字节吗,即使大多数实际数据项的成本更低超过 10 个字节?

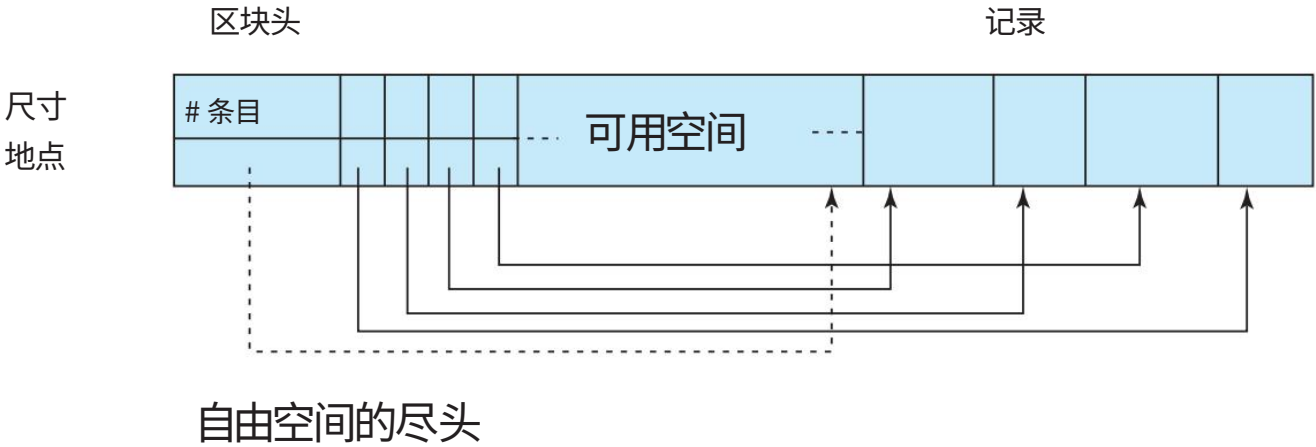
# 可变长度记录

- 属性按顺序存储
- 可变长度属性由固定大小（偏移量、长度）表示,实际数据存储在所有固定长度属性之后
- 空值由空值位图表示



# 块中的可变长度记录

- 开槽页眉包含：
  - 记录条目数
  - 块中的可用空间已结束
  - 每个记录的位置和大小
  - 可以在页面内移动记录,以保持记录连续且没有空  
它们之间有空格;必须更新标题中的条目
- 指针不应该直接指向记录      相反,它们应该指向头文件中记录的条目
- 页面大小通常与磁盘块大小一致 (4KB-8KB)



# 存储大对象

- 例如, BLOB/CLOB 类型

- BLOB: 二进制大对象
- CLOB: 字符大对

- 象
- 记录必须小于页面
- 替代方案:

- 在文件系统中存储为文件
- 将文件名 (通常

是文件系统中的路径) 存储为记录的属性  
数据库

- 通过 B+ 树存储为由数据库管理的文件

- B+-tree 文件组织允许我们读取整个对象, 或指定的字节范围  
对象, 以及插入和删除对象的部分
- 分解成碎片并存储在单独关系中的多个元组中
- 

PostgreSQL TOAST



# 文件中记录的组织

- 堆**– 只要有空间,记录就可以放置在文件中的任何地方。
- 顺序**– 根据记录的值按顺序存储记录。

每条记录的搜索关键字

- 多表聚类文件组织**

- 可以将几种不同关系的记录存储在同一个文件中
- 动机:将相关记录存储在同一个块上,以最大限度地减少 I/O

- B+-树文件组织**

- 有序存储,即使有插入/删除

- 散列** 根据搜索键计算的散列函数;结果指定记录应该放在文件的哪个块中

# 堆文件组织

- 记录可以放置在文件中任何有可用空间的地方
  - 记录一旦分配通常就不会移动

- 能够有效地找到文件内的可用空间非常重要 ·可用空间图

·每个块有 1 个条目的数组。每个条目占一个字节的几个位,记录空闲块的比例 ·在以下示例中,每个块有 3 个位,值除以 8 表示空闲块的比例

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- 可以拥有二级自由空间地图
  - 在下面的例子中,每个条目最多存储 4 个第一级自由空间映射条目

4	7	2	6
---	---	---	---

- 定期将空闲空间映射写入磁盘,某些条目有错误 (旧)值是可以的 (将被检测并修复)

# 顺序文件组织

- 根据每个搜索键的值按顺序存储记录  
记录
- 适用于需要顺序处理整个文件的应用程序
- 文件中的记录按搜索键排序

10101 斯里尼瓦桑	计算机科学	65000	
12121 吴	金融	90000	
15151 莫扎特	音乐	40000	
22222 爱因斯坦	物理史物	95000	
32343 埃尔赛义德	理学计算	60000	
33456 黄金	机科学	87000	
45565 卡茨		75000	
58583 预选赛	历史 金融	62000	
76543 辛格		80000	
76766 克里克	生物学计	72000	
83821 布兰特	算机科学	92000	
98345 金	电气。工程师。	80000	



# 顺序文件组织

·在插入和删除记录时很难保持物理顺序

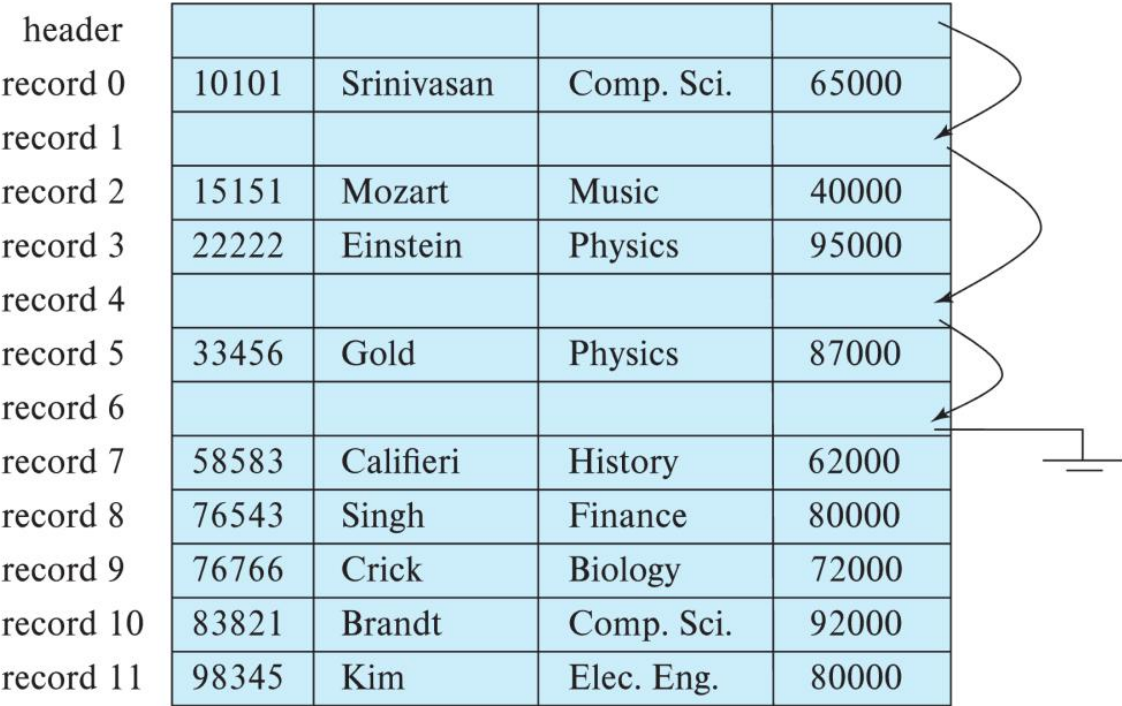
·删除 – 使用指针链 ·插入 – 找到

记录将被插入

·如果有可用空间,则插入那里 ·如果没有可用空间,则将记录插入溢出  
堵塞

·无论哪种情况,都必须更新指针链 ·需要不时重新组

织文件  
恢复顺序



# 多表聚类文件组织

- 使用多表聚类文件组织将多个关系存储在一个文件中
  - 适用于涉及以下内容的查询：
    - 部门讲师
    - 或：一个部门及其教师（一个对多对应关系）
  - 不适合仅涉及部门的查询
    - 例如，选择来自部门
  - 产生可变大小的记录
    - 可以添加指针链来链接特定关系的记录
  - 不适合大型数据库
    - 需要除连接之外的其他操作

部门

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

讲师

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

部门和讲师的多表聚类,聚类键是 dept\_name

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

# 分区

- 表分区:关系中的记录可以划分为较小的关系,并单独存储
  - 例如,交易关系可以分为transaction\_2018、 transaction\_2019等。
- 事务中编写的查询必须访问所有分区中的记录
  - 除非查询有 year=2019 之类的选择,这种情况下只需要一个分区
- 分区
  - 降低某些操作的成本,例如可用空间管理
    - 由于某些操作的成本（例如为记录查找可用空间)会随着关系的增加而增加尺寸
  - 允许将不同的分区存储在不同的存储设备上
    - 例如,当前年份的交易分区放在 SSD 上,较早年份的交易分区放在磁盘上

# 列式存储

- 也称为柱状表示

- 分别存储关系的每个属性

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000



# 列式存储

·好处:

·如果只需要某些属性,则减少 IO ·提高 CPU 缓存性能

·数据分析查询通常连续访问一个属性的多个值 ·提高压缩率 ·同一类型的数据可以更有效  
地压缩

·现代 CPU 架构上的矢量处理 ·缺点

·从列表示重建元组的成本 ·元组删除和更新的成本 ·解压缩的成本

\_\_\_\_\_

\_\_\_\_\_



# 列式存储

- 发现列式表示比行式表示更有利于决策支持
- 传统的面向行的表示法更适合事务加工
- 一些数据库支持两种表示形式
  - 称为混合行/列存储

# 列式存储

- ORC (优化行列式)文件格式 行导向表示转换为

列导向表示 每个条带包含列子集的值

- 文件格式为文件内部采用列式存储

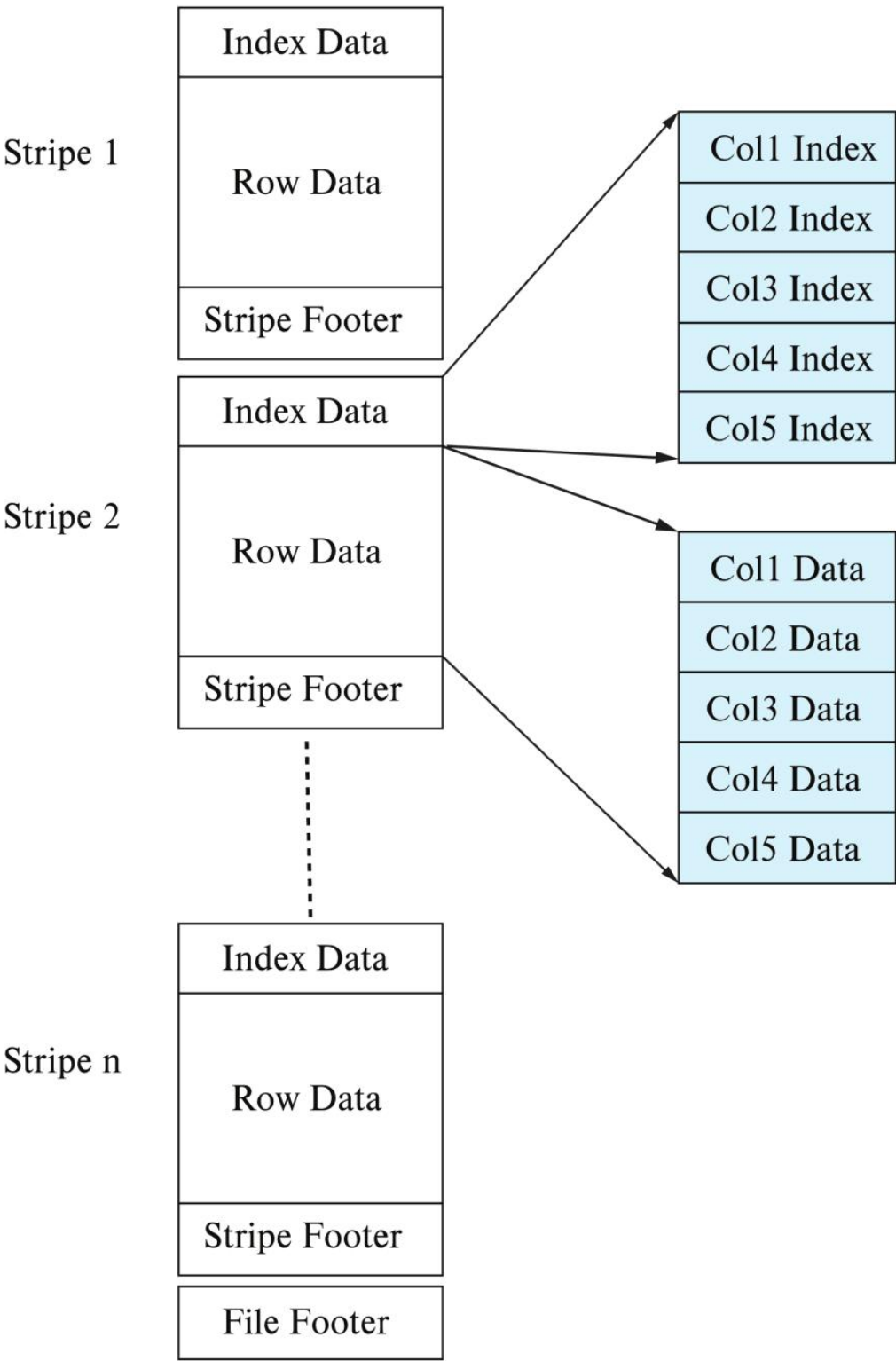
- 行数据

- 索引数据

- 如果您对大数据处理感兴趣,请参阅以下详细信息:

[cwiki.apache.org/](https://cwiki.apache.org/)

[confluence/display/hive/languagemanual+orc](https://confluence/display/hive/languagemanual+orc)



# 数据字典存储

·数据字典（也称为系统目录）存储元数据……即有关数据的数据 “meta”表示“的”

- 

·包括

- 有关关系的信息

- 关系名称

- 每个关系的属性的名称、类型和长度
  - 视图的名称和定义

- 完整性约束
  - 用户和会计信息

- 为每个用户授权,用户的默认模式,密码等
  - 统计和描述数据

- 每个关系中的元组数量
  - 每个属性的不同值的数量

# 数据字典存储

- 包括（继续）
  - 物理文件组织信息
    - 关系如何存储（顺序/哈希/...）
    - 关系的物理位置
    - 有关

索引/索引的信息



（我们稍后会学习）

# 系统元数据的关系表示

· 我们可以使用专用数据结构和代码来存储元数据 但更好的方法是将它们存储为

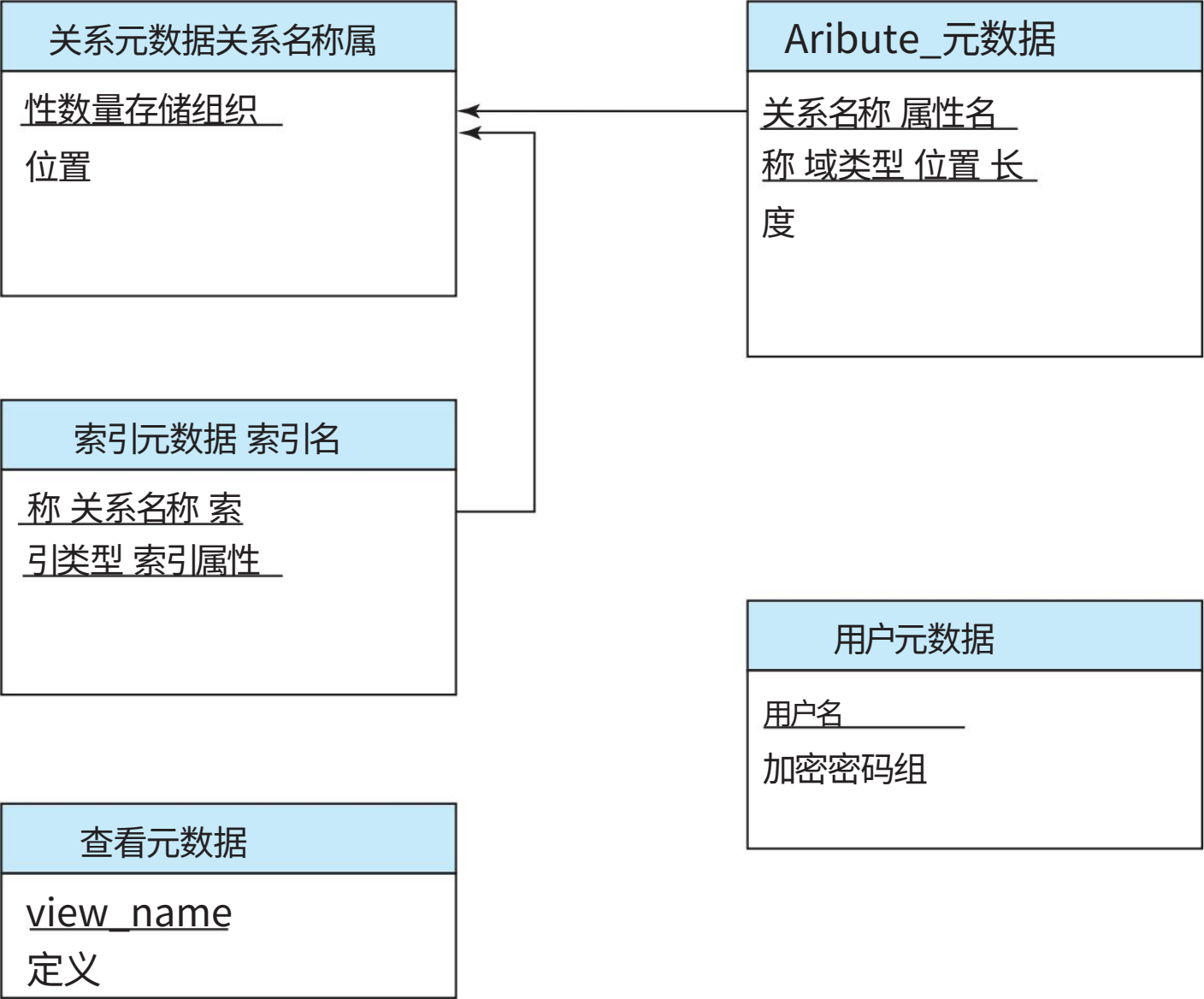
数据库中的关系

· 关系表示的示例  
元数据

· 磁盘上的关系表示 为高效访问而设计的内存中

的专门数据结构 不同的 DBMS 可能有自己的实现 首先查  
阅元数据,然后获取记录 也许将元数

据放在内存中以实现快速访问更好



# 存储访问

- 许多大型数据库的容量远远超过服务器上的可用内存

  - 大多数数据库中的数据库数据主要驻留在磁盘上

  - 必须将数据载入内存才能读取或更新;并且更新后的数据随后必须将块写回到磁盘

- 块是存储分配和数据传输的单位

- 数据库系统力求最大限度地减少磁盘和内存

  - 我们可以通过在磁盘中保留尽可能多的块来减少磁盘访问次数  
主存储器

  - 缓冲区:主内存中用于存储磁盘块副本的部分

  - 缓冲区管理器:负责在主内存中分配缓冲区空间的子系统

# 自学

· 数据库系统概念，第 13.5 章 “数 第七版  
数据库缓冲区”