第4章: 存储管理

Storage Management

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件: znzou@hit.edu.cn

2023年秋

Outline[1]

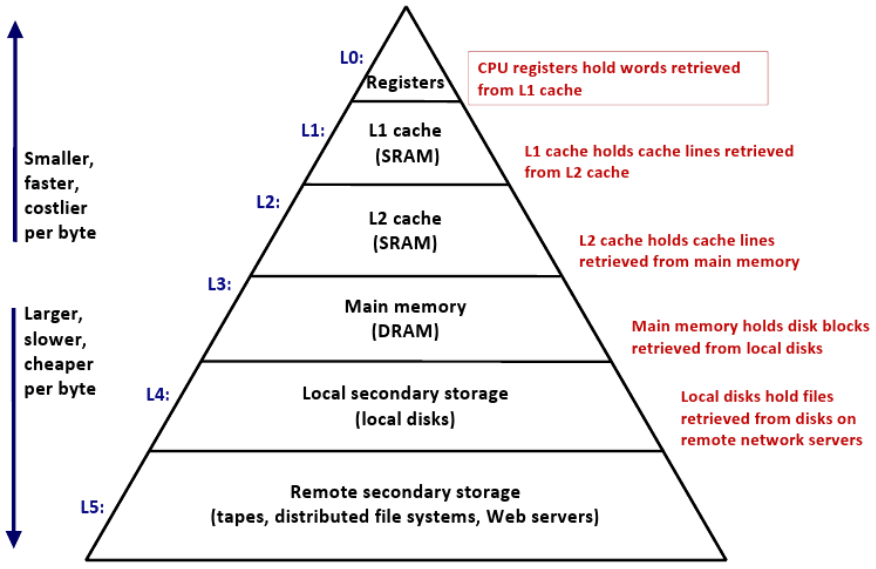
[1] 更新于2023年10月23日

Storage Media

存储层级(The Memory Hierarchy)

计算机系统的存储器被组织成层次结构(hierarchy)

- 越靠上层的存储器离CPU越近，容量越小，速度越快，每字节价格越高
- 越靠下层的存储器离CPU越远，容量越大，速度越慢，每字节价格越低



存储器的分类(一)

按CPU访问存储介质的方式，可将存储器分为三类

- 主存储器(primary storage)
- 二级存储器(secondary storage)
- 三级存储器(tertiary storage)

主存储器(Primary Storage)

主存储器包括:

- 寄存器(register)
- 高速缓存(cache)
- 内存(main memory)

主存储器的特点

- 主存储器是按字节寻址的(byte-addressable)
- CPU可使用`load/store`指令直接访问主存储器中的数据

二级存储器(Secondary Storage)

二级存储器包括:

- 磁盘(magnetic disk)/机械硬盘(hard disk drive, HDD)
- 闪存(flash memory)/固态硬盘(solid state drive, SSD)

二级存储器的特点

- 二级存储器是按块寻址的(block-addressable)
- 二级存储器是联机(online)使用的
- CPU无法直接访问二级存储器中的数据
- CPU若要访问二级存储器中的数据，必须使用`read/write`指令将数据先从二级存储器复制到主存储器

三级存储器(Tertiary Storage)

三级存储器包括:

- 磁带(magnetic tape)
- 光盘(optical disk)
- 网络存储(network storage)

三级存储器的特点

- 三级存储器是按块寻址的(block-addressable)
- 三级存储器是脱机(offline)使用的
- CPU若要访问三级存储器中的数据，必须先将数据从三级存储器复制到二级存储器

访存时间(Access Time)

访存时间/ns	存储介质
0.5	L1 cache
7	L2 cache
100	DRAM
150,000	SSD
10,000,000	HDD
30,000,000	网络存储
1,000,000,000	磁带

存储层次之间的数据传输

- Cache
  ↕ 单位: 缓存行(cache line), 大小: 64B
- DRAM
  ↕ 单位: 块(block)/页(page), 大小: 512B–16KB
- 二级存储器
  ↕ 单位: 块(block)/页(page), 大小: 512B–16KB
- 三级存储器

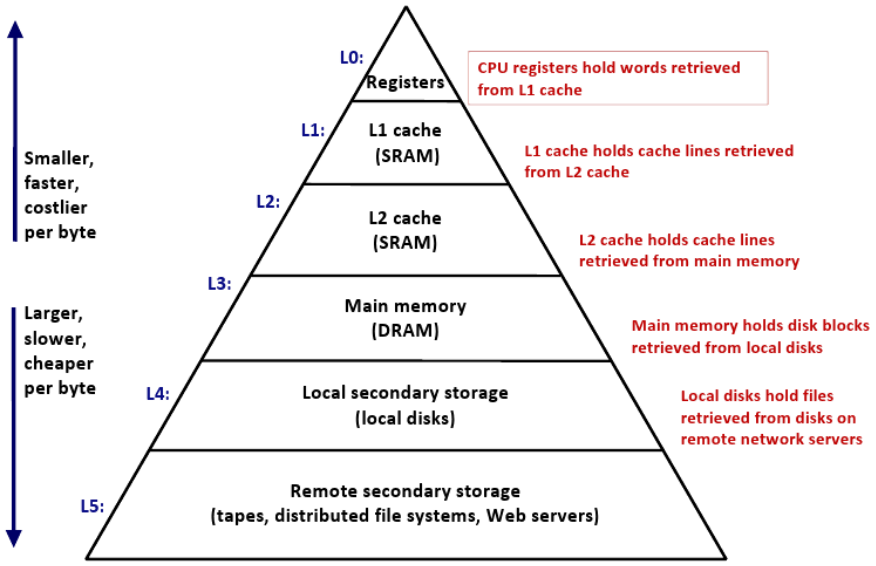
数据局部性(Data Locality)

同一单元中的数据经常同时被访问

虚拟内存(Virtual Memory)

虚拟内存使应用程序认为它拥有连续可用的内存

- 虚拟内存不是存储层级中的一层



存储器的分类(二)

按存储介质的易失性(volatility)/持久性(persistence)，可将存储器分为:

- 易失性存储器(volatile storage): 计算机重启后，易失性存储器中的数据会丢失
  - 主存储器
- 非易失性存储器(non-volatile storage): 计算机重启后，非易失性存储器中的数据不会丢失(计算机系统故障损坏存储器的情况除外)
  - 二级存储器
  - 三级存储器

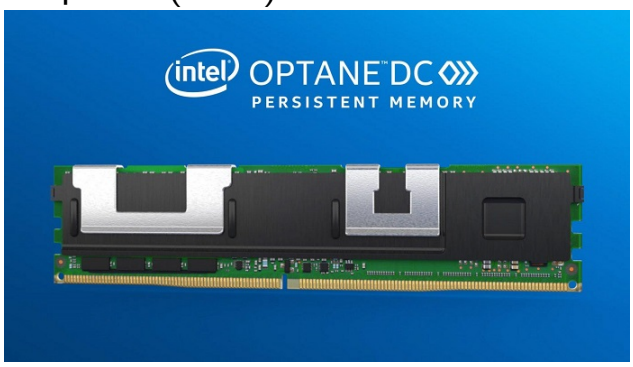
存储器	寻址方式	易失性
主存储器	按字节寻址	易失
二级存储器	按块寻址	非易失
三级存储器	按块寻址	非易失
???	按字节寻址	非易失

持久性内存(Persistent Memory, PMem)

持久性内存又称非易失性内存(non-volatile memory, NVM)

- 按字节寻址
- 非易失

Intel Optane (傲腾) DC Persistent Memory[2]



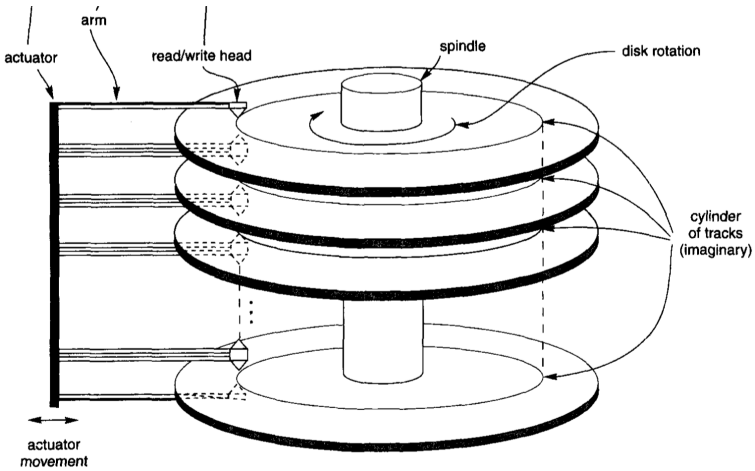
Persistent Memory Developing Toolkit (PMDK), `http://pmem.io`

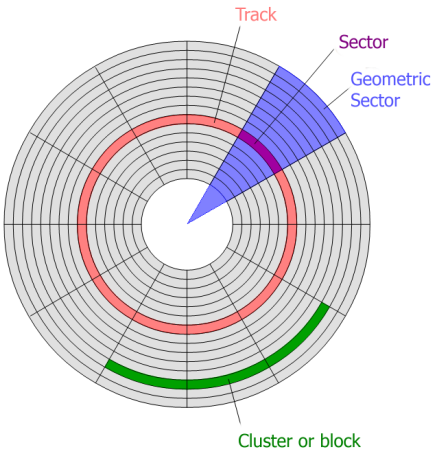
[2] `https://www.intel.cn/content/www/cn/zh/architecture-and-technology/optane-dc-persistent-memory.html`

磁盘(Magnetic Disk)

磁盘由两部分构成

- 磁盘构件(disk assembly): 扇区(sector) ⊂ 磁道(track) ⊂ 柱面(cylinder)
- 磁头构件(head assembly): 磁头(disk head)和磁臂(disk arm)





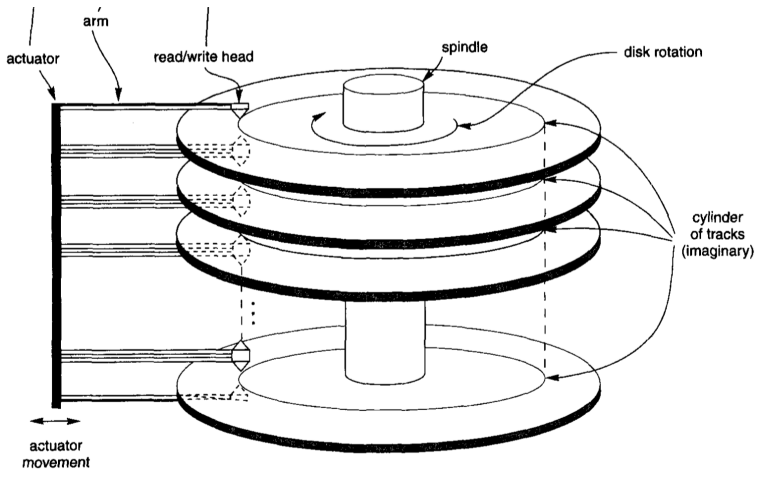
磁盘块(Block)/页(Page)

操作系统在格式化磁盘时将磁盘划分为许多大小相同的块(页)

- 磁盘块(页)的大小是扇区大小(通常512B)的整数倍
- 磁盘块(页)的大小一旦被设置，则不可以动态改变
- 磁盘块(页)的大小通常是512B–16KB

磁盘控制器(Disk Controller)

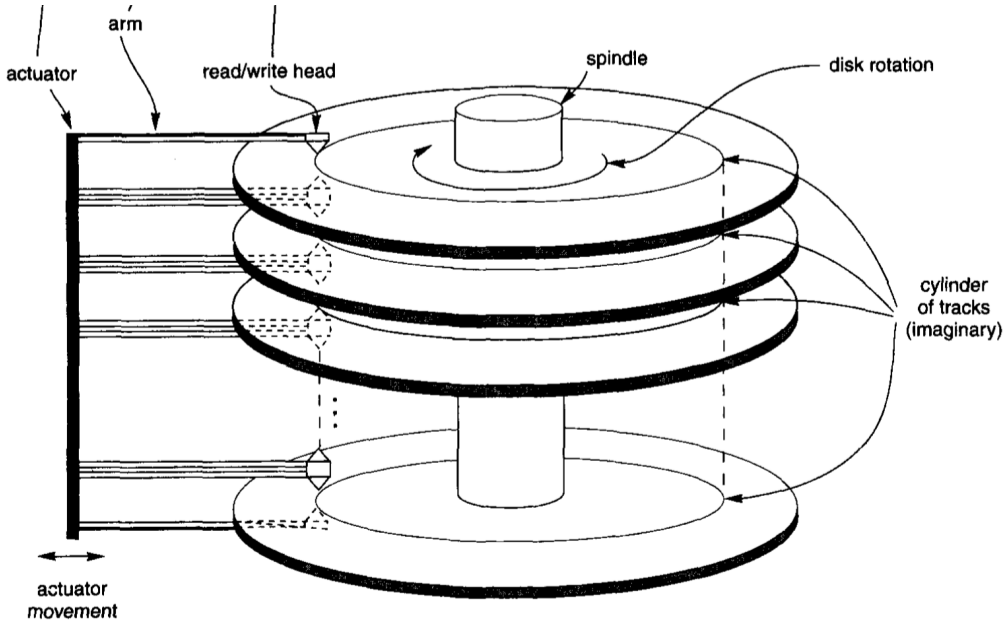
1. 寻道: 寻找目标磁道
2. 旋转: 寻找目标扇区
3. 传输: 在主存储器与磁盘之间传输数据
4. 预取: 读取相同磁道或柱面上相邻的磁盘块(只有读第一块时产生寻道时间和旋转延迟)



磁盘延迟

磁盘延迟= 寻道时间+ 旋转延迟+ 块传输时间≈ 10 ms

1. 寻道时间(seek time) ≈ 0–10 ms
2. 旋转延迟(rotational latency) ≈ 0–10 ms
3. 块传输时间(block transfer time) < 1 ms



I/O请求

DBMS ⟶ OS ⟶ 磁盘控制器

每秒I/O操作次数(I/O Operations per Second, IOPS): 衡量磁盘访问效率

固态硬盘(Solid-State Drive, SSD)

构造

- 使用NAND闪存(Flash Memory)作为存储介质
- 具有与磁盘相同的面向块的接口

特点

- 随机读写速度高于磁盘
- 功耗低于磁盘

SSD的写操作

特点

- 先擦除，后写入
- 擦除块通常是256KB–1MB

缺点

- 写放大(write amplification)
- 寿命低: 擦除次数100,000–1,000,000次

解决方案

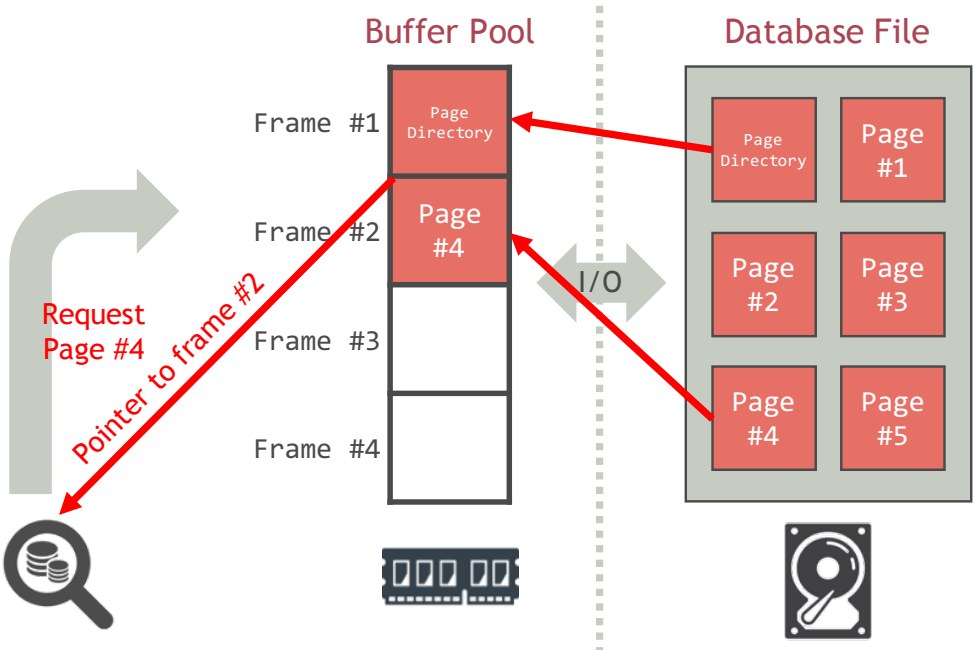
- 逻辑页号$\longrightarrow$物理页号

Disk-Oriented Database Storage

面向磁盘的数据库存储

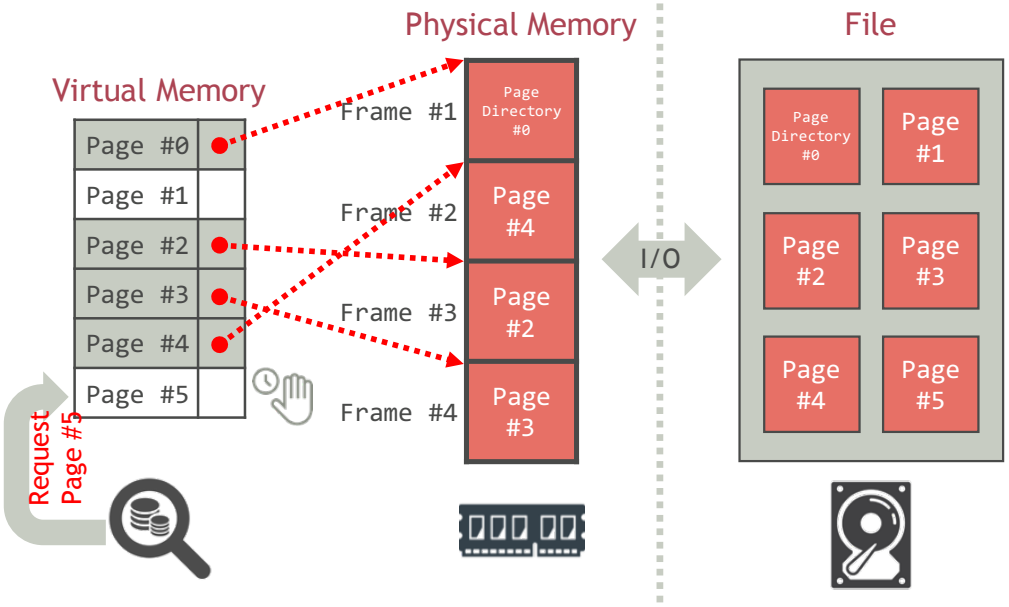
将数据库持久存储在磁盘文件中

- 读/写数据前需要将文件页从磁盘读入内存
- 写数据后需要将修改过的文件页写回磁盘，替换原有页



为何不用操作系统直接管理数据库存储?

- 使用操作系统的内存映射(memory mapping, mmap)将数据库文件的内容映射到进程的地址空间
- 操作系统负责在磁盘和内存之间移动文件页面



为何该方法不适用于数据库存储管理?

设计目标

- 使DBMS能够管理比可用内存容量更大的数据库
- 避免不合理的磁盘I/O造成的系统停顿(stall)和性能下降

空间方面的设计

- Q: 将页写到磁盘中什么位置?
- A: 尽量将常在一起使用的页在物理上相近地存储(数据局部性)

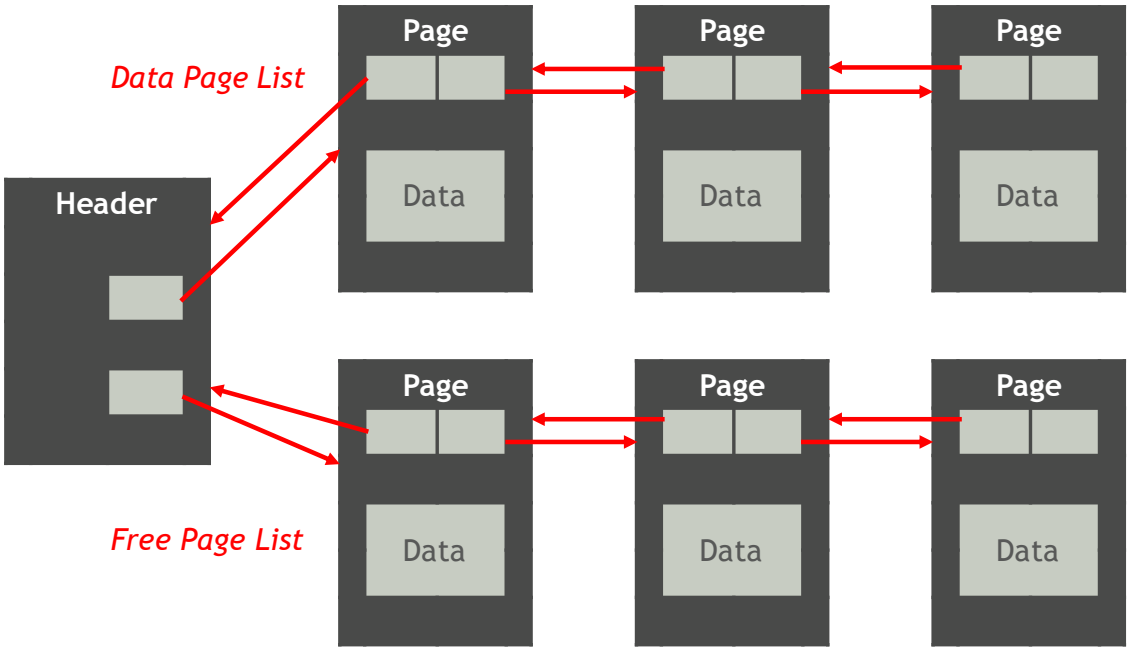
时间方面的设计

- Q: 何时将页读入内存? 何时将页写回磁盘?
- A: 最小化磁盘I/O造成的系统停顿次数

Disk-Oriented Database Storage
File Storage

数据库的文件存储(File Storage)

- 将一个数据库存储为一个或多个文件(file)
- 每个文件包含多个页(page)



数据库文件的页

数据库文件的页中可以存储元组、元数据、索引、日志记录等

- 大多数DBMS在一个页中只存储一种类型对象
- 页的大小通常是512B–16KB 演示

每个页拥有唯一的编号，称作页号(page ID)

- DBMS使用间接层(indirection layer)将页号映射为页的物理地址

存储管理器(Storage Manager)

DBMS的存储管理器负责管理数据库文件

- 记录页中元组的读/写
- 记录页中的空闲空间

操作系统不了解数据库文件的内容

文件存储的分类

- 面向行的存储(row-oriented storage)
- 面向列的存储(column-oriented storage)

Row-Oriented Storage

Row-Oriented Storage
Value Representation

数的表示

INT/INTEGER/SMALLINT/BIGINT型整数

- 采用C/C++的整数表示

FLOAT(n)/REAL/DOUBLE PRECISION型浮点数

- 采用IEEE-754标准的浮点数表示
- 通常比定点数运算快
- 存在舍入误差

NUMERIC(p,s)/DECIMAL(p,s)型定点数

- 表示为可变长二进制序列(含元数据)
- 不存在舍入误差

字符串的表示

CHAR(n)/CHARACTER(n)型定长字符串

- 表示为n个字符的数组
- 如果字符串长度小于n，则数组中字符串后面的字符用空字符(null)补全
- 例：'cat'用CHAR(5)型字符串表示为

'c'	'a'	't'	0	0

VARCHAR(n)/CHARACTER VARYING(n)/CLOB型变长字符串

- 不使用C/C++中以空字符结尾的字符串表示
- 数组头部若干字节存储字符串的长度，随后存储字符串的内容
- 例：'cat'用VARCHAR(5)型字符串表示为

3	'c'	'a'	't'

字节串的表示

BINARY(n)型定长字节串

- 表示为n个字节的数组
- 如果字节串长度小于n，则数组中字节串后面的字节用0补全

VARBINARY(n)/BINARY VARYING(n)/BLOB型变长字节串

- 表示为可变长字节数组
- 数组头部若干字节存储字节串的长度，随后存储字节串的内容

Row-Oriented Storage
Tuple Layout

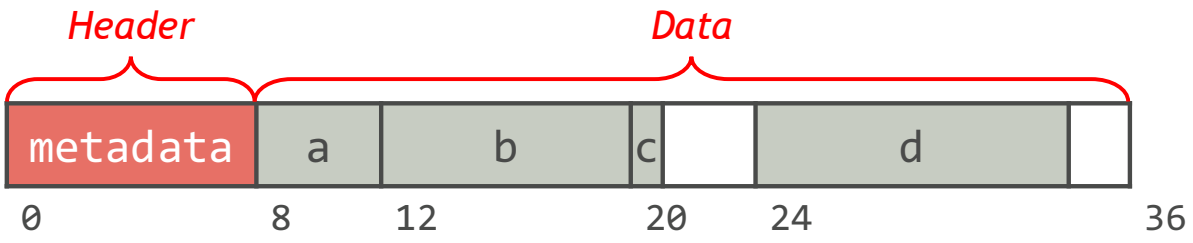
元组的表示(Tuple Layout)

元组表示为字节序列(a sequence of bytes)

- DBMS根据元组所在关系的模式(schema)，将元组的字节序列表示翻译为元组的全部属性值
- DBMS的系统目录(catalog)记录关系的模式定义

元组表示包含两部分

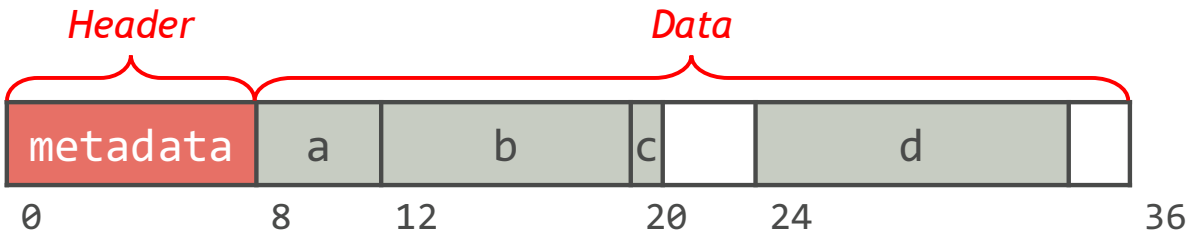
- 元组头(tuple header)
- 元组数据(tuple data)



元组头(Tuple Header)

元组头记录元组的元数据(meta-data)

- 指向元组所在关系的模式定义的指针
- 元组的长度
- 元组的最后修改时间
- 元组的可见性(与并发控制相关)
- 元组的哪些属性值非空，用位图(bitmap)表示

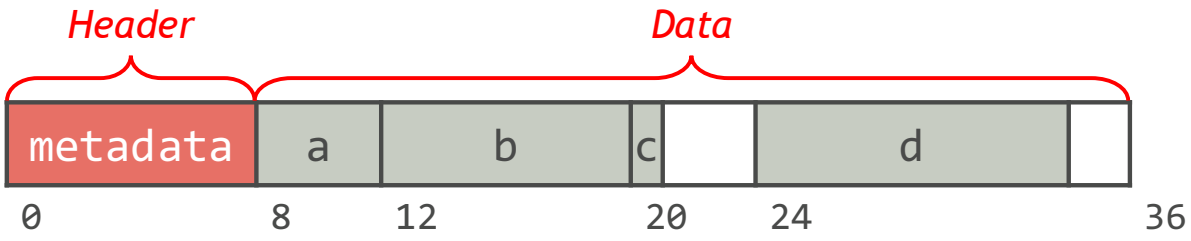


元组数据(Tuple Data)

元组的数据部分由元组的所有属性值拼接而成

- 通常按定义关系模式时指定的属性顺序存储属性值
- 每个属性值在元组中存储位置的偏移量是4字节或8字节的倍数

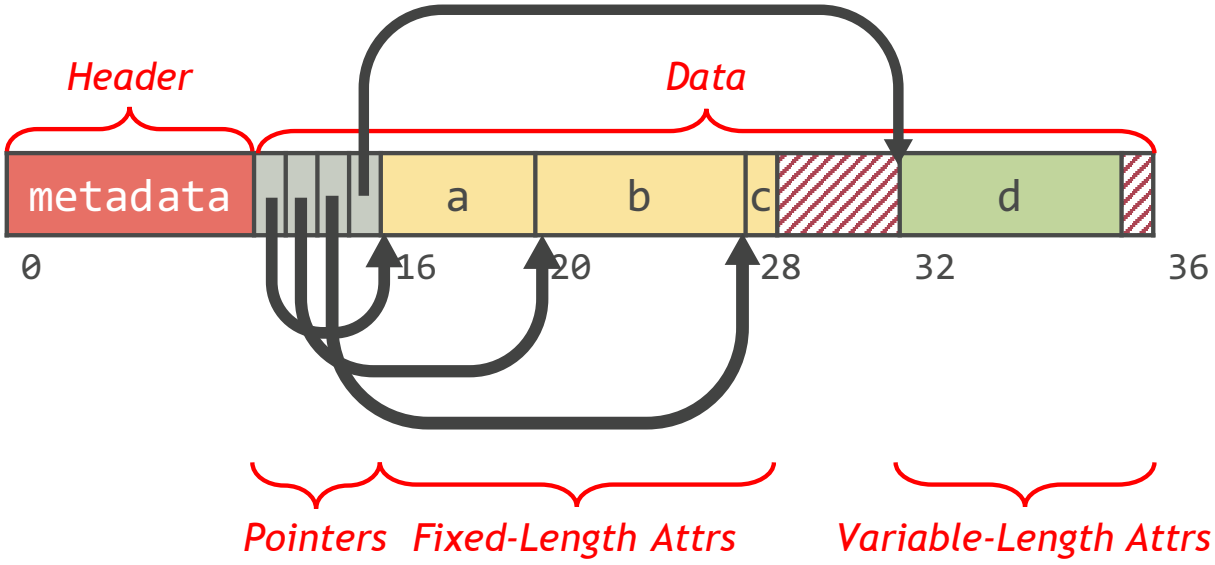
```
CREATE TABLE Foo (
  a INT NOT NULL,
  b BIGINT NOT NULL,
  c CHAR NOT NULL,
  d VARCHAR(10) NOT NULL);
```



变长元组的布局

定长属性值和变长属性值分别置于元组两端

- 元组头后紧跟指针数组，指向每个属性值

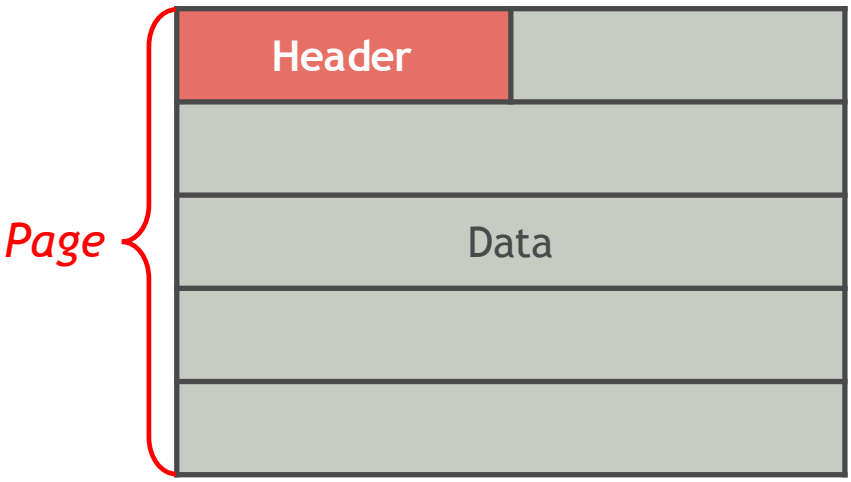


Row-Oriented Storage Page Layout

页布局(Page Layout)

一个页包括两部分

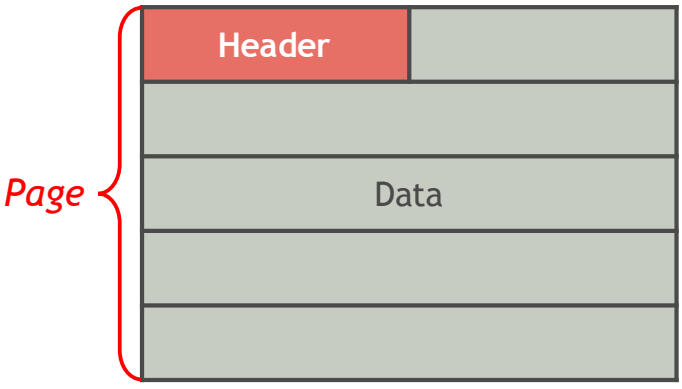
- 页头(page header)
- 页数据(page data)



页头(Page Header)

页头记录页的元数据

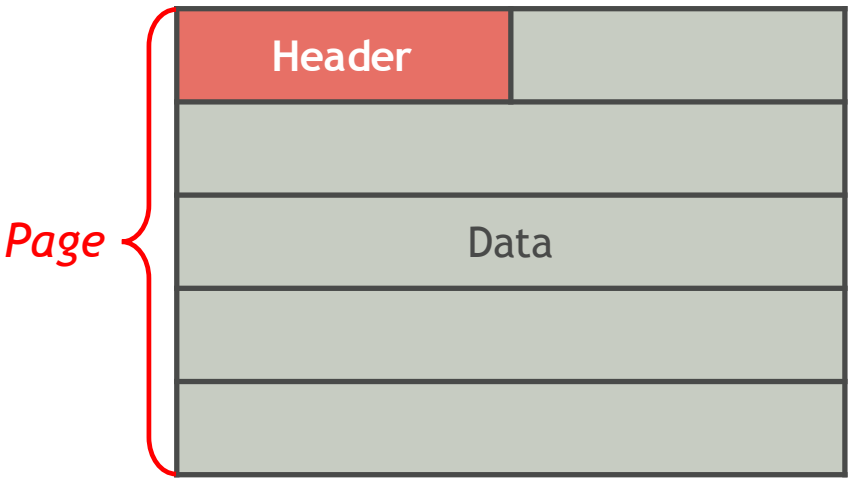
- 页的大小
- 页的校验和(checksum)
- DBMS的版本
- 页的可见性(与并发控制相关)
- 页中数据的数据压缩方法



页数据(Page Data)

页中数据的组织方法(以存储元组的页为例)

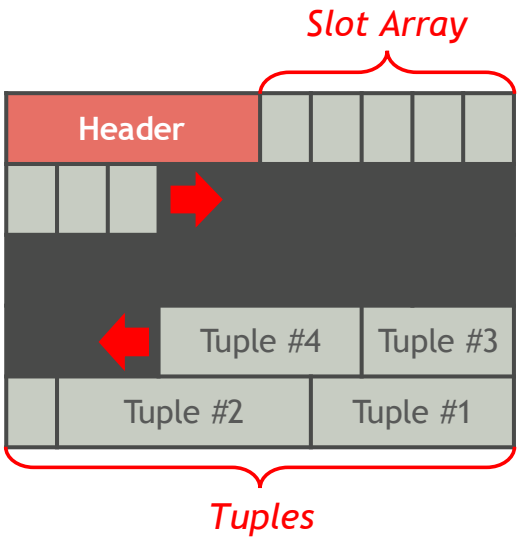
- 面向元组(tuple-oriented)的组织方法
- 日志结构(log-structured)的组织方法



面向元组的页数据组织方法

分槽页(slotted page)是最常见的面向元组的页数据组织方法

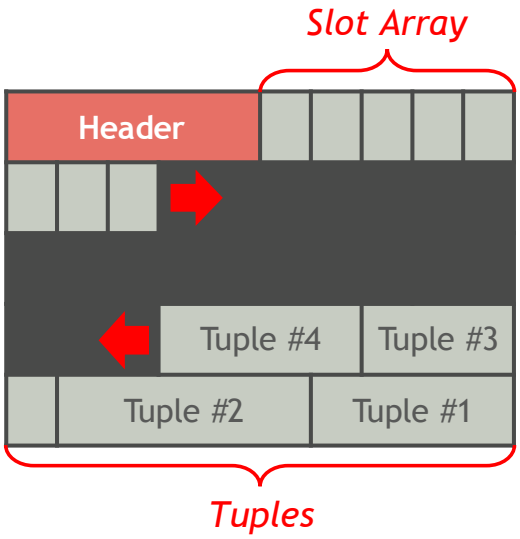
- 槽数组(slot array)
- 元组序列



分槽页(Slotted Page)

每个元组占一个槽(slot)

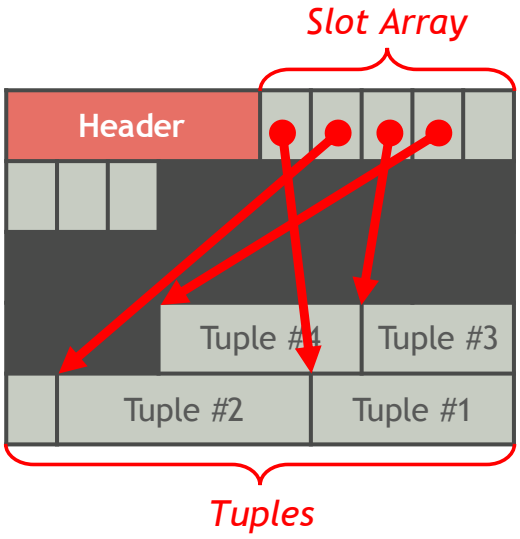
- 槽从后向前布置在页的末尾
- 每个槽的起始位置的偏移量是4字节或8字节的倍数



分槽页(续)

页头后放置槽数组(slot array)

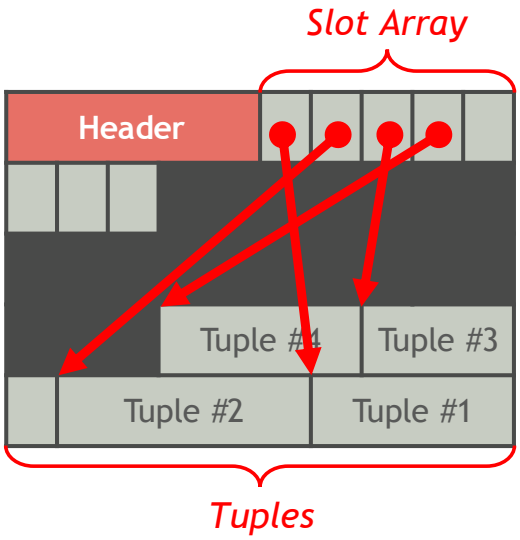
- 槽数组的第i个非空元素存储第i个槽的起始位置的偏移量



槽的元数据

槽的元数据存储在页头中

- 槽的数量: 用于确定下一个空闲槽的编号
- 最后一个槽的起始位置的偏移量: 用于确定新元组的插入位置



记录号(Record ID)

DBMS为一个关系中的每个元组分配唯一的记录号(record ID)

方法1: 使用(页号, 槽号)作为记录号 演示

- 页号(page ID): 包含元组的页的编号
- 槽号(slot number): 元组在页中的槽的编号

方法2: 用唯一的整数作为记录号

- DBMS使用间接层(indirection layer)将元组ID映射为(页号, 槽号)

溢出页(Overflow Page)

- DBMS基本不允许元组的大小超过页的大小
- 如果一个元组的大小超过页的大小，DBMS可将元组中CLOB/BLOB型属性值存储在溢出页中，并在元组中存储指向溢出页中CLOB/BLOB型大对象的指针

思考题

使用溢出页有什么坏处?

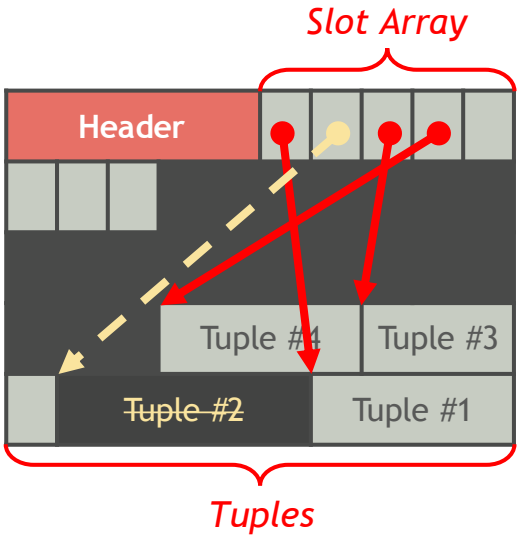
页碎片化(Page Fragmentation)

随着元组删除或版本过期，页中会产生碎片(fragment)

- 浪费磁盘空间
- 增加磁盘I/O

应定期回收空间

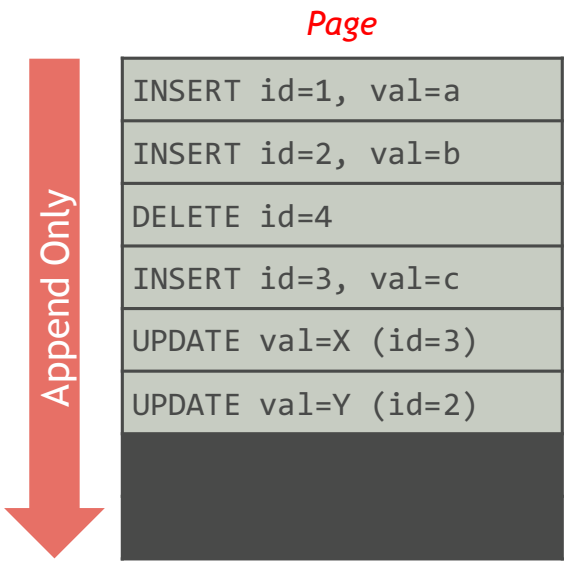
- VACUUM命令 ▸ 演示



日志结构页布局(Log-Structured Page Layout)

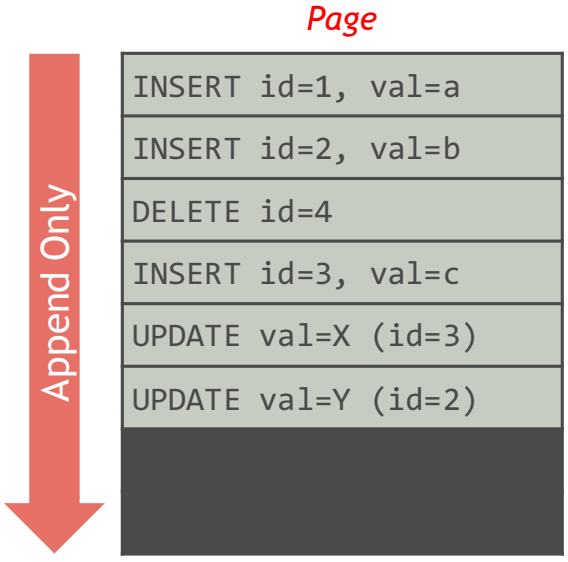
文件中只存储数据更新操作的日志记录(log record)，而不存储元组

- 新的日志记录只能写到文件末尾



日志记录(Log Record)

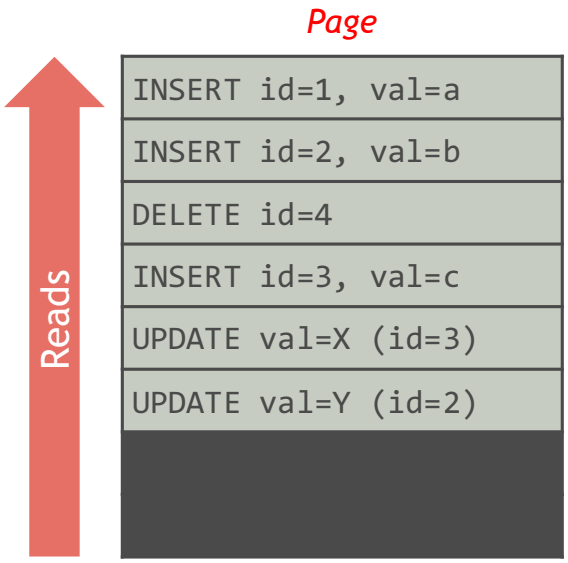
- 插入操作的日志记录包含整个插入的元组
- 删除操作的日志记录包含被删除的元组的主键
- 修改操作的日志记录包含被修改的元组的主键，被修改的属性及修改后的属性值



读元组

按主键查找元组的最新属性值

- 从后向前扫描文件中的日志记录
- "重建"元组的最新属性值

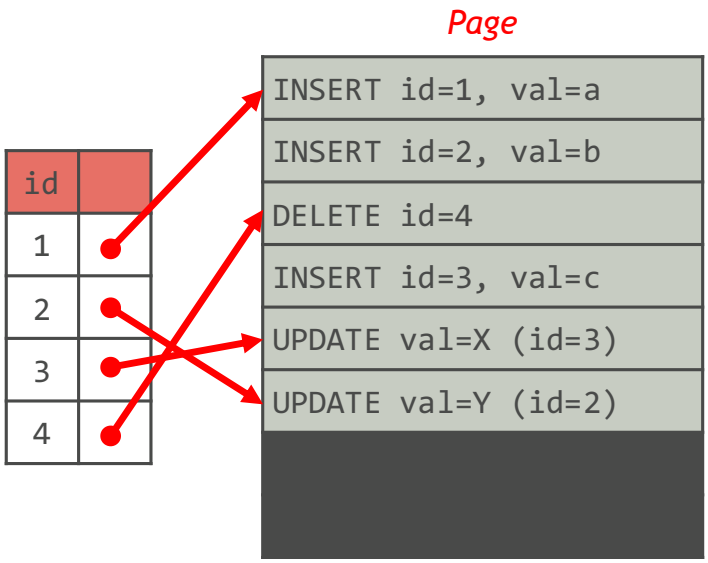


(id=1, val=a); (id=2, val=Y); (id=3, val=X); id=4 deleted;

日志记录索引

为了加快读元组的速度，可以在多个连续的页(run)上建立索引

- 将元组的主键映射为页中与该元组相关的最后一条日志记录的位置



日志记录合并(Compaction)

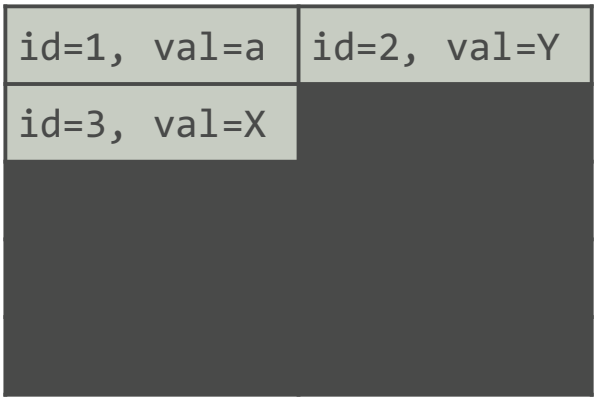
一条元组在文件中存在大量相关的日志记录

- 空间开销大
- 读放大(read amplification): 为了读一条元组，必须进行多次I/O

合并(compact)文件中的日志记录

- 写放大(write amplification): 更新一条元组后可能引发合并，需要进行多次I/O

Compacted Page



Row-Oriented Storage
File Organization

文件组织

不同的DBMS采用不同的方法组织文件中的页

- **方法1:** 堆文件组织(heap file organization)
- **方法2:** 顺序/有序文件组织(sequential/sorted file organization)
- **方法3:** 哈希文件组织(hash file organization)

堆文件组织(Heap File Organization)

堆文件(heap file)中的元组以任意顺序存储

堆文件管理

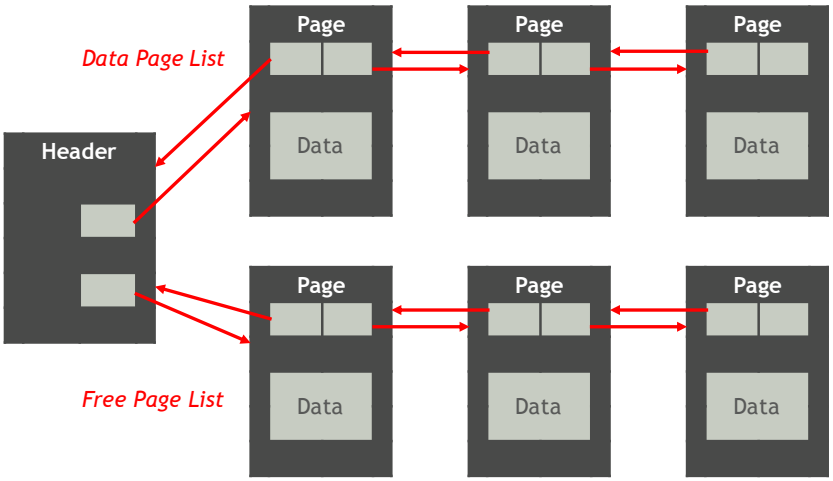
- 创建页、读页、修改页、删除页
- 遍历堆文件的所有页

堆文件中页的组织方法

- 基于链表(linked list)的组织方法
- 基于页目录(page directory)的组织方法

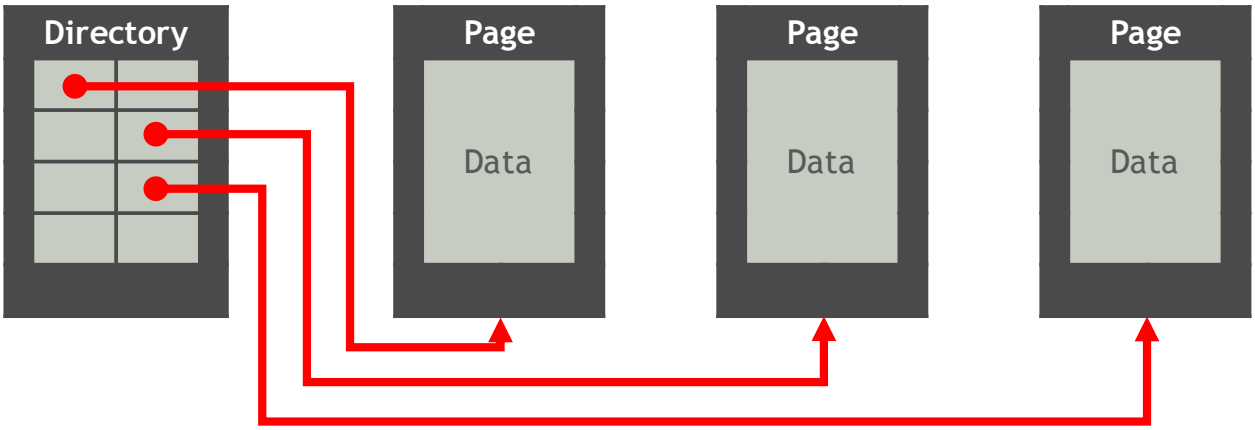
基于链表的页组织方法

- 数据页(data page): 存储元组，所有数据页组织成链表
- 空闲页(free page): 所有空闲页组织成链表
- 头页(header page): 存储在文件的起始位置，记录堆文件的元数据、指向数据页链表头的指针、指向空闲页链表头的指针



基于页目录的堆文件页组织方法

- 数据页(data page): 存储元组
- 页目录(page directory): 记录每个数据页的位置和空闲空间信息
- DBMS必须保证页目录中的信息与数据页的状态同步



顺序/有序文件组织(Sequential/Sorted File Organization)

顺序/有序文件中的元组按排序键的顺序存储

- 除非排序键是关系的主键，否则很少使用顺序/有序文件

思考题

顺序/有序文件有什么缺点?

哈希文件组织(Hash File Organization)

- DBMS使用一个哈希函数
- DBMS根据元组键的哈希值来确定元组存储在哈希文件的哪个页
- 具体实现参考下一章介绍的外存哈希表

Metadata Storage

元数据(Metadata)存储

除了存储数据外，DBMS还要存储元数据(meta-data)，又称数据字典(data dictionary)或系统目录(system catalog)

- 关系模式
- 索引、视图等信息
- 用户、授权等信息
- 关系与属性的统计信息等

常用的DBMS通常把元数据存储在自己的数据库中

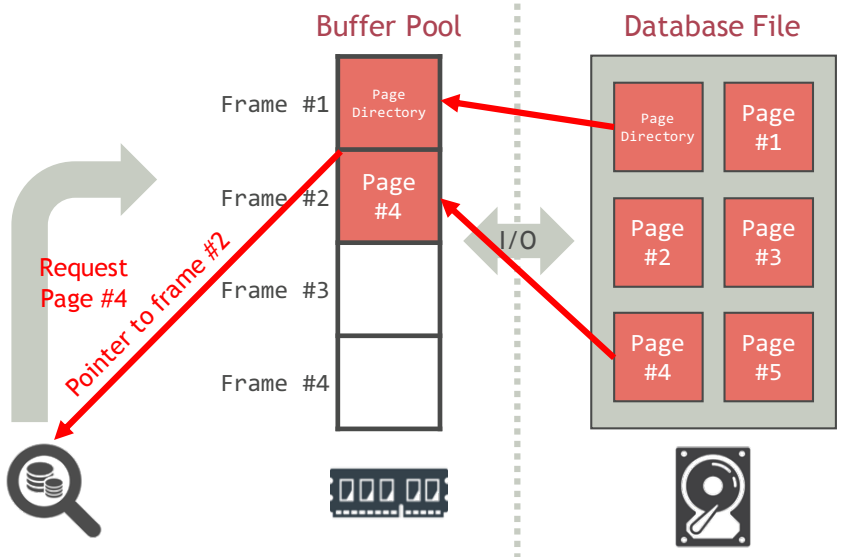
Example

PostgreSQL: `\l \d \dt \di \dv`
MySQL: `show, describe`

Buffer Management

缓冲区管理(Buffer Management)

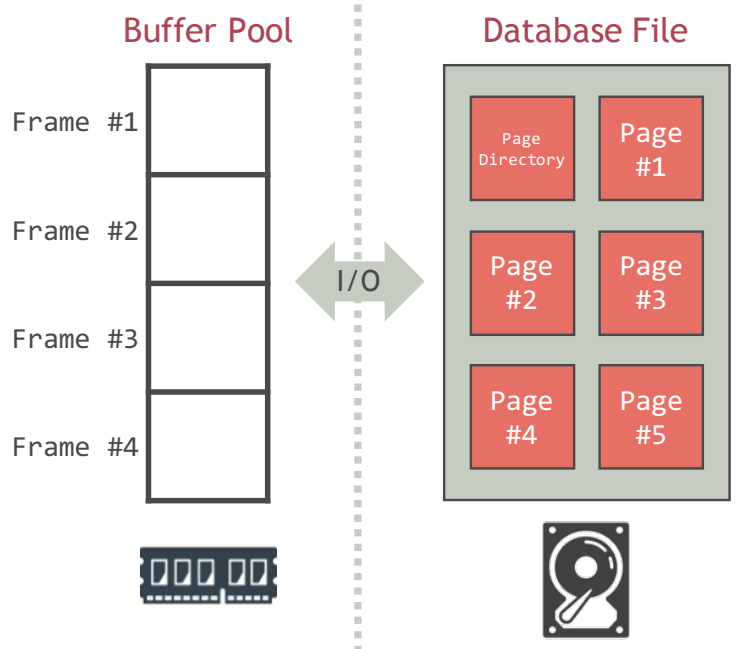
- CPU无法直接读/写磁盘中的数据，必须先将文件中的页从磁盘读入内存(缓冲区)
- 数据库文件的大小经常超过DBMS的可用内存容量
- 缓冲区管理器负责在磁盘和内存之间复制文件页



缓冲池(Buffer Pool)

DBMS将可用内存区域划分为页数组，称作缓冲池

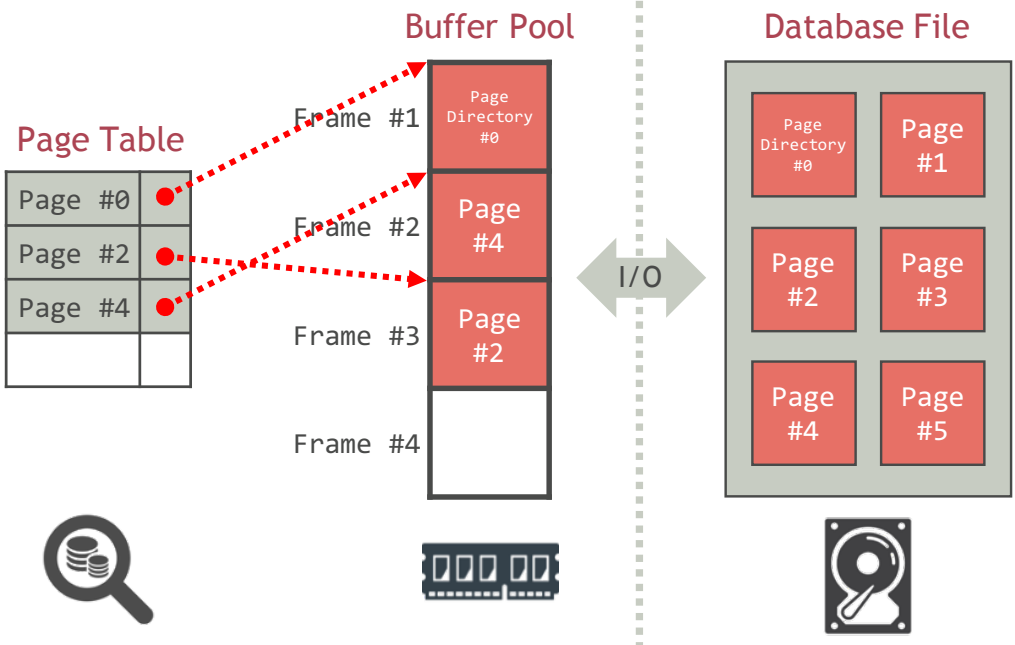
- 缓冲池中的页称作页框(frame)



缓冲池的设计: 页表(Page Table)

页表(page table)记录缓冲池中当前有哪些页以及这些页在内存中的地址

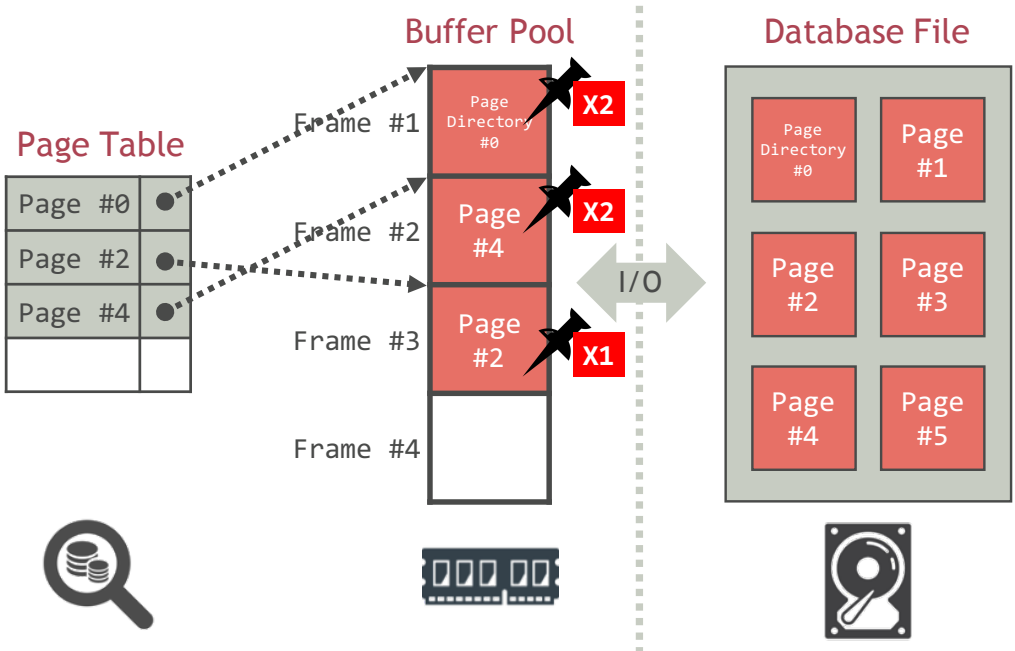
- 页表将页号(page ID)映射为该页所在页框的地址
- 页表可以用内存哈希表实现



缓冲池的设计: 页框的元数据

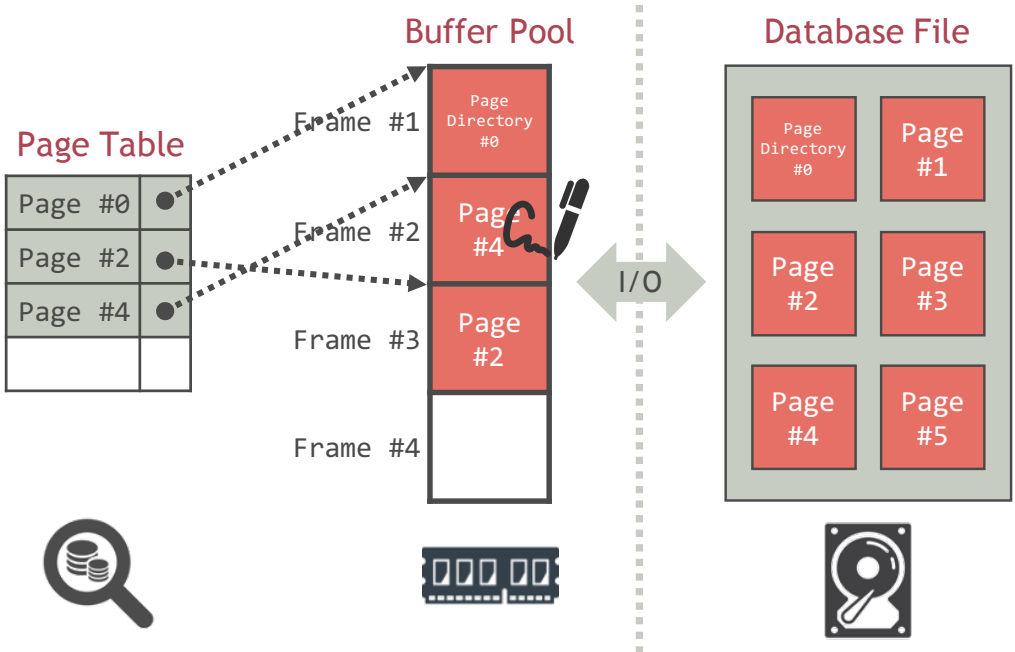
DBMS用两个变量记录缓冲池中每个页框的状态

- pin_count: 页框中的页当前被请求但未释放的次数，即引用计数



缓冲池的设计: 页框的元数据(续)

- dirty: 自从页框中的页被读入缓冲池后，该页是否被修改过

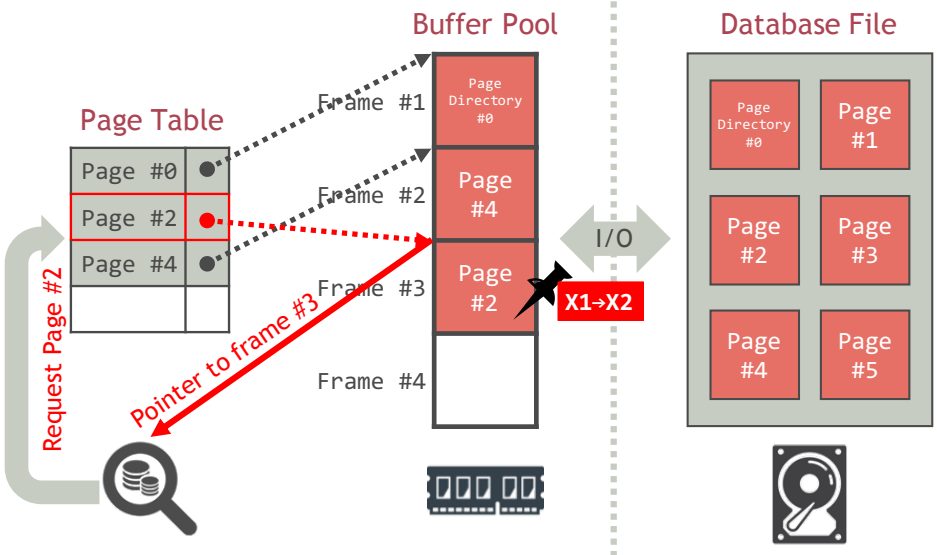


缓冲区管理器的功能

- 请求页(requesting pages)
- 修改页(modifying pages)
- 释放页(releasing pages)

请求页: 情况1

1. 搜索页表，检查缓冲池中是否存在被请求的页P
2. 如果P在缓冲池中，则钉住(pin)P，即将包含P的页框的pin_count加1
3. 返回包含P的页框的地址

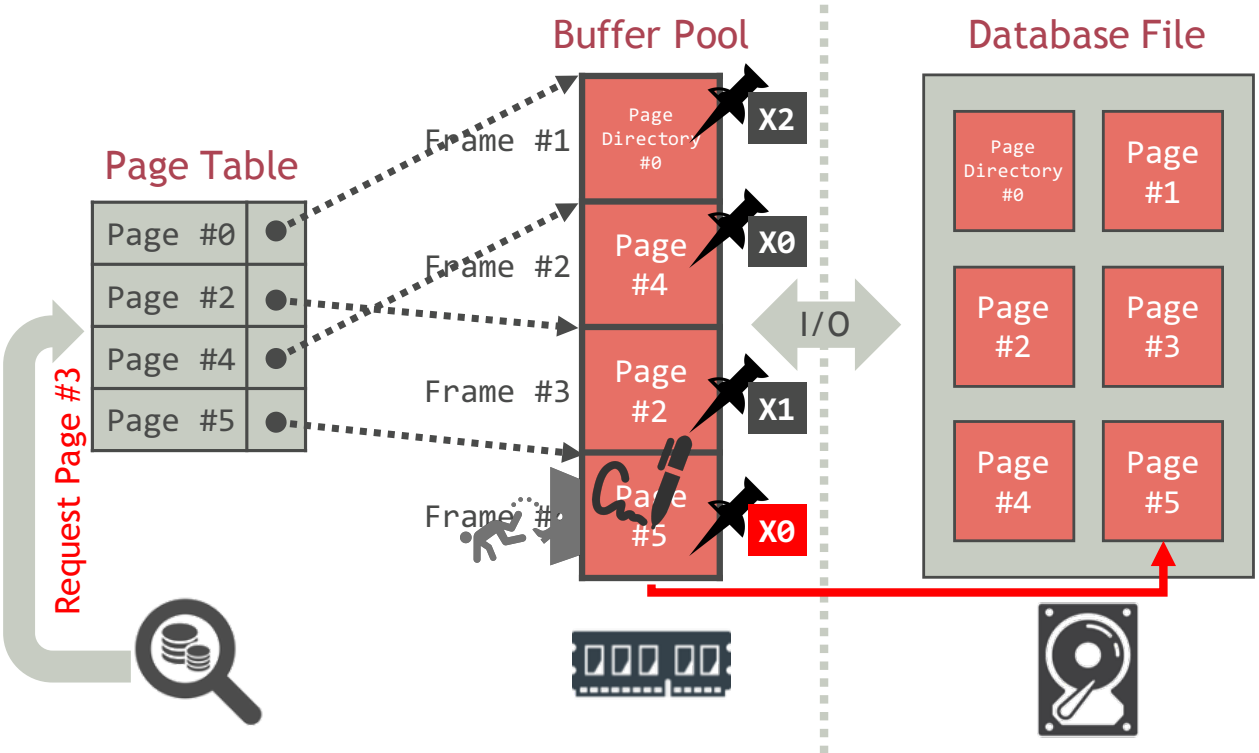


请求页：情况2

如果请求的页P不在缓冲池中，则执行下列步骤

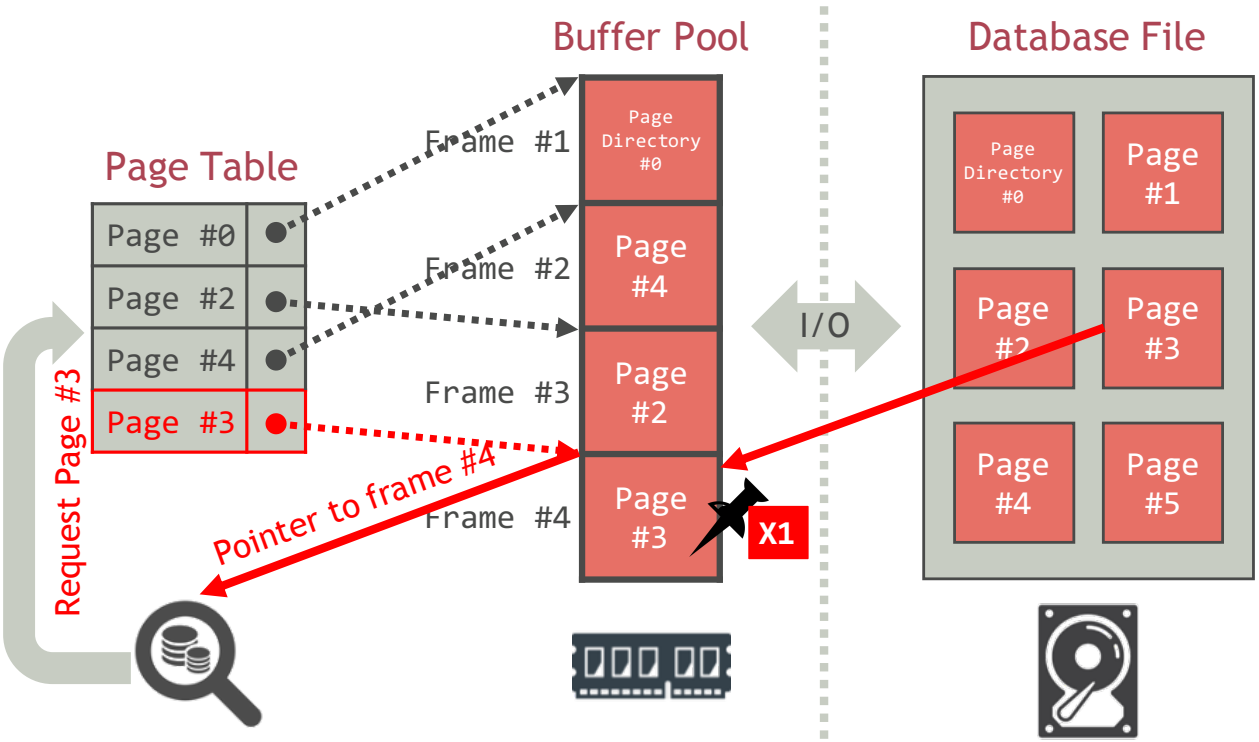
1. **选替换页**：使用缓冲池的页替换策略(page replacement policy)，从缓冲池中选择一个pin_count = 0的页框，将该页框的pin_count加1
2. **写回脏页**：如果该页框的dirty值为真，则将该页框中的页写回磁盘文件
3. **读请求页**：将请求的页P读入该页框，返回该页框的地址

请求页：情况2(续)



页替换

请求页：情况2(续)

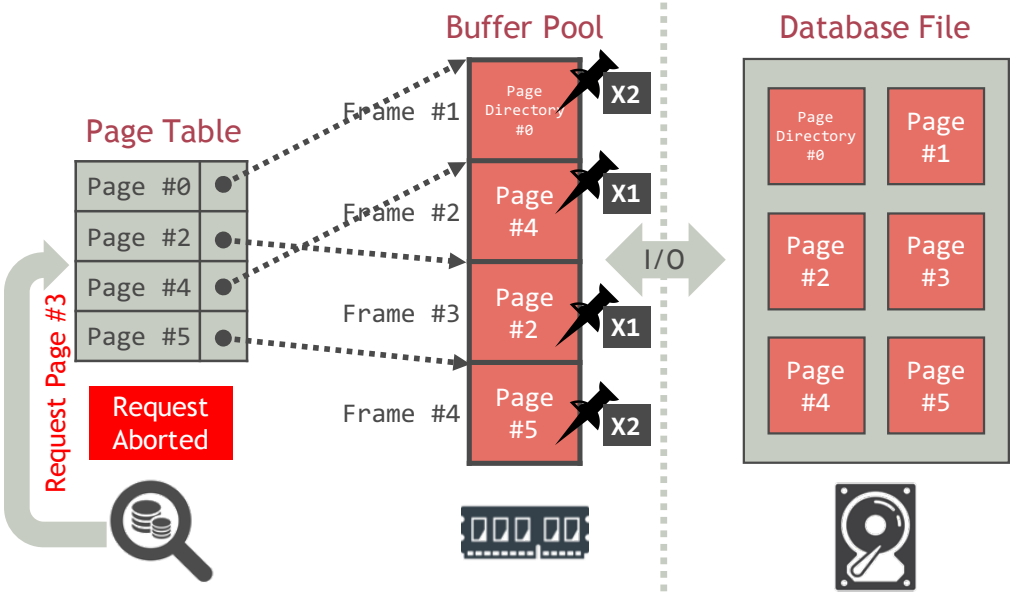


读请求页

请求页：情况3

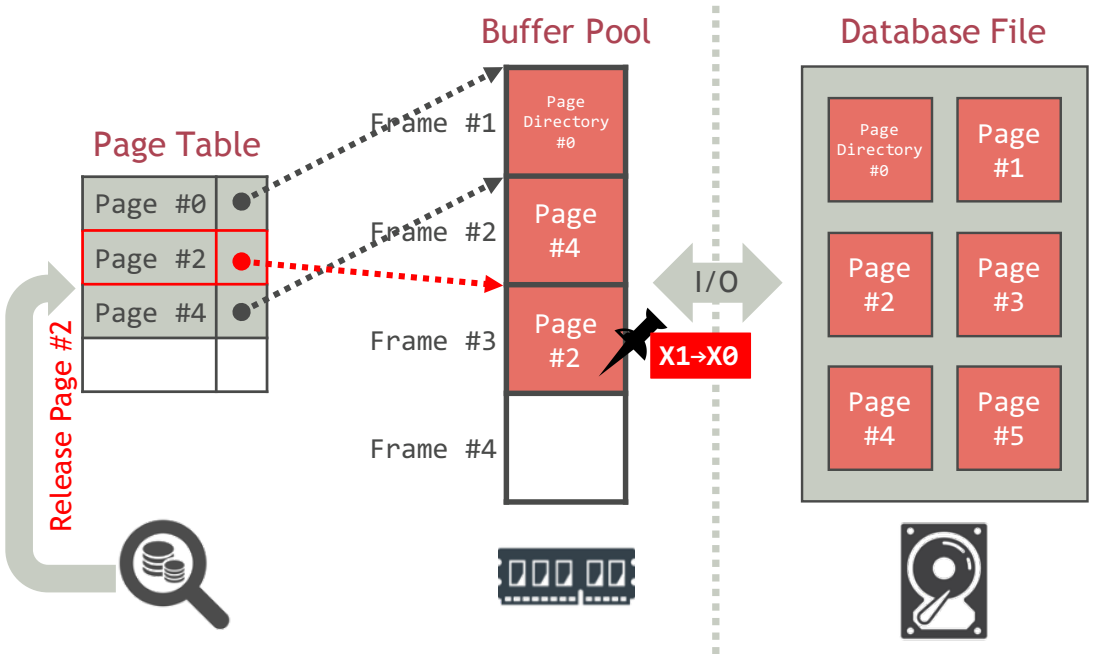
如果缓冲池中没有pin_count = 0的页框，则请求开始等待，直至DBMS释放了缓冲池中的页面

- 实际上，DBMS会终止(abort)提出该请求的事务，并重新执行它



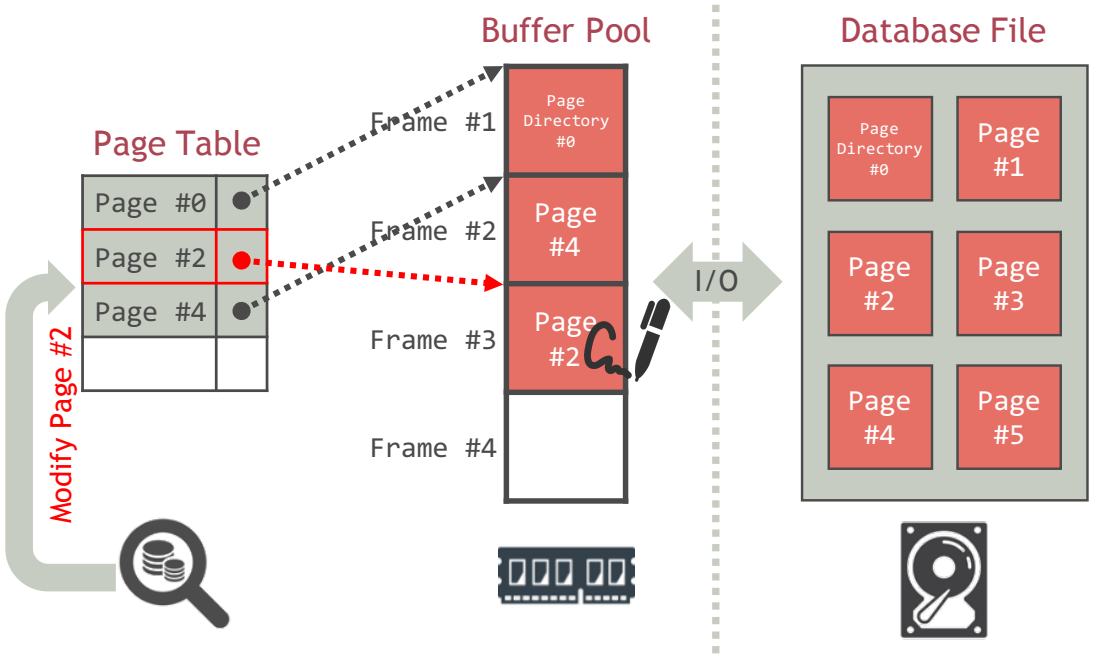
释放页

Unpin被释放的页，即将包含该页的页框的pin_count减1



修改页

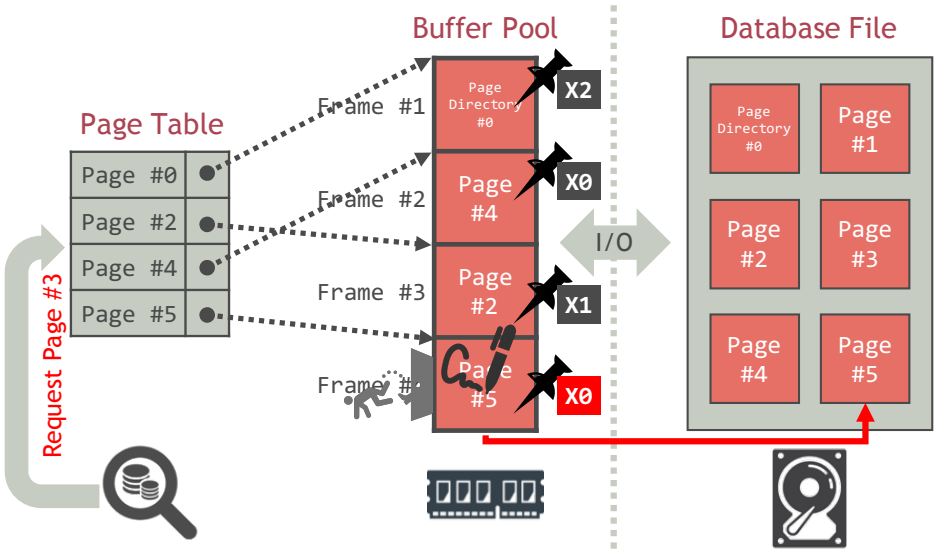
将被修改的页所在的页框的dirty值置为真



页替换策略(Page Replacement Policies)

页替换策略对数据库操作的时间影响很大

- 最近最少使用策略(Least Recently Used, LRU)
- 时钟替换策略(Clock)
- 先进先出策略(First-In First-Out, FIFO)
- 最近最多使用策略(Most Recently Used, MRU)



最近最少使用策略(Least Recently Used, LRU)

运行时协议

- 用一个时间戳记录缓冲池中每个页框的最后访问时间
- 为了提高效率，可将缓冲池中pin_count = 0的页框按时间戳排序

页替换协议

- 替换pin_count = 0且时间戳最旧的页框中的页

思考题

LRU策略有什么缺点?

时钟替换策略(Clock Replacement Policy)

时钟替换策略是对LRU策略的近似

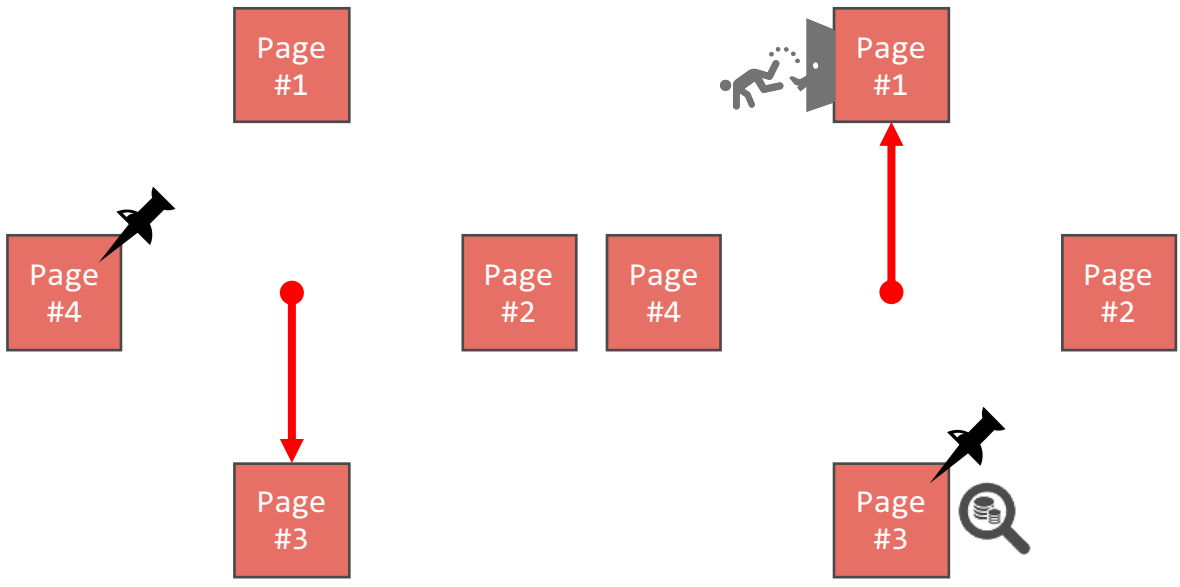
运行时协议

- 将缓冲池中所有页框组织成一个环形缓冲区(表盘)
- 为每个页框设置一个引用位(reference bit)
- 初始时，将所有页框的引用位都置为0
- 当一个页被读入一个页框时，将该页框的引用位置为1
- 当一个页框中的页被访问时，将该页框的引用位置为1

时钟替换策略(续)

页替换协议

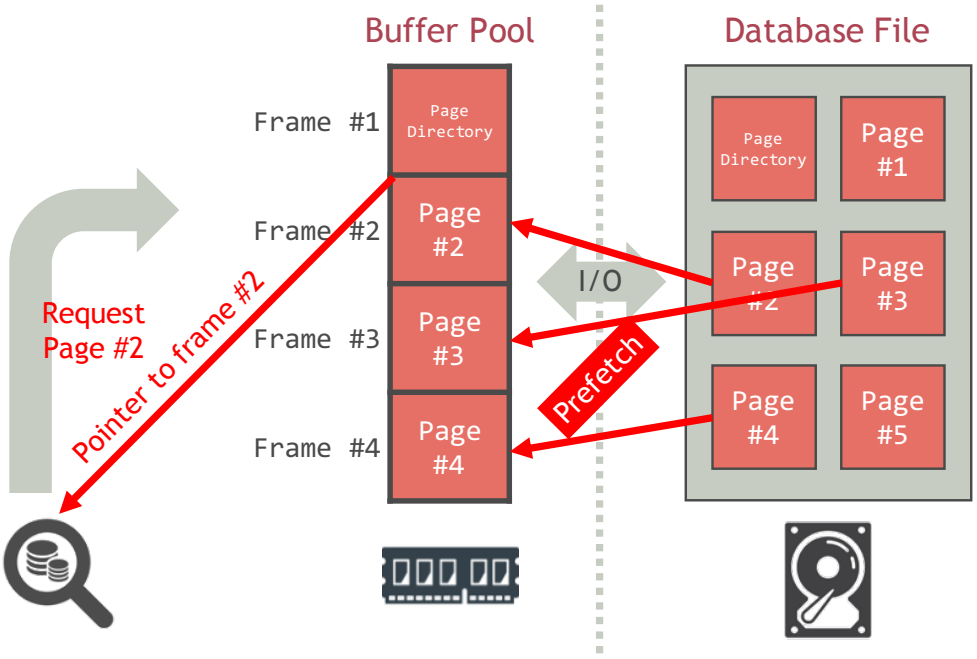
- 用一根旋转的表针搜索首个pin_count = 0且引用位为0的页框，替换该页框中的页
- 如果表针扫过页框的引用位为1，则将其引用位置为0



预取(Prefetching)

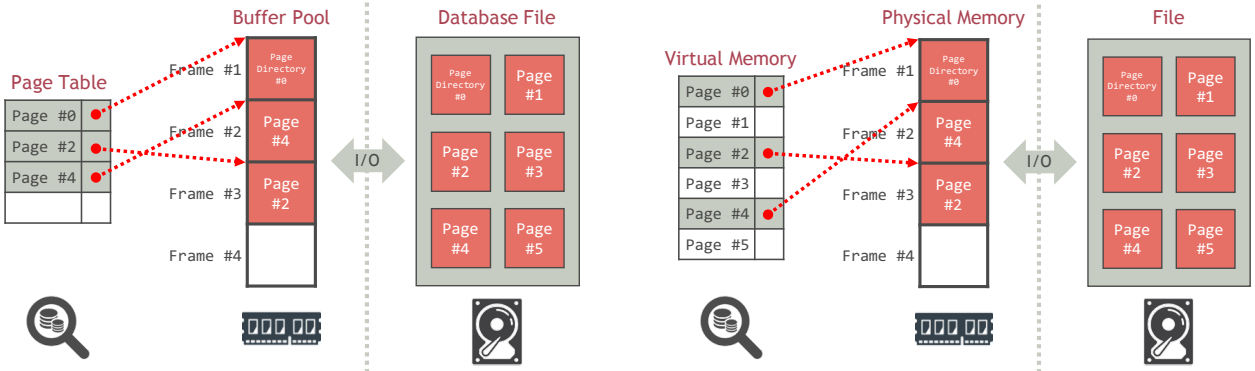
缓冲区管理器经常从文件中预取一些可能会在未来被请求的页

- 如果被预取的页真的被请求了，则页请求会很快完成



缓冲池与虚拟内存的相同点

- 两者都用于访问可用内存容纳不下的数据
- 两者都会在需要时将页从磁盘读入内存
- 当需要内存中的空闲页时，两者都会进行页替换



思考题

为什么不基于操作系统的虚拟内存构建DBMS呢?

缓冲池与虚拟内存的不同点1

和OS相比，DBMS经常能够更准确地预测页的访问顺序

- 有助于更好地选择被替换的页(该页很可能不会在短时间内再次被请求)
- 有助于更准确地预取页(被预取的页很可能在短时间内被请求)

缓冲池与虚拟内存的不同点2

OS不具备将内存中的页强制(force)写回磁盘的能力；而DBMS必须具备强制写回的能力

- OS会将修改过的页延迟(defer)写回磁盘。如果计算机系统在将页写回磁盘前或在写回过程中出现故障，则会破坏数据的一致性
- 基于缓冲区管理器的强制写回能力，可以实现预写式日志(Write-Ahead Logging, WAL)协议(见故障恢复)，确保在将修改过的页写回磁盘前已将日志记录强制写入日志文件，从而保证故障恢复

并发控制(concurrency control)和故障恢复(failure recovery)使缓冲区管理器的设计变得更加复杂

Column-Oriented Storage

面向行的存储(Row-Oriented Storage)

面向行的存储将一个元组的所有属性存储在一条记录中

- 当查询仅涉及少量属性时，也必须读无关属性值，浪费I/O
- 缓存中相邻字节可能包含不需要的属性值，缓存效率低
- 将不同类型的属性值存储在一条记录中，压缩率低

Example (面向行的存储)

CS-001	Elsa	F	19	CS
CS-002	Ed	M	19	CS
MA-001	Abby	F	18	Math
PH-001	Nick	M	20	Physics

```
SELECT Sdept, AVG(Sage) FROM Student GROUP BY Sdept;
```

面向列的存储(Column-Oriented Storage)/柱状存储(Columnar Storage)

面向列的存储将关系的每个属性单独存储

- 当查询仅涉及少量属性时，不需要读无关属性，减少I/O
- 缓存中相邻字节来自于同一个属性，缓存效率高
- 相同类型的属性值存储在一条记录中，压缩率高
- 可以使用SIMD指令执行向量化计算

Example (面向行的存储/柱状存储)

CS-001	Elsa	F	19	CS
CS-002	Ed	M	19	CS
MA-001	Abby	F	18	Math
PH-001	Nick	M	20	Physics

```
SELECT Sdept, AVG(Sage) FROM Student GROUP BY Sdept;
```


面向列的存储的缺点

面向列的存储适合于联机分析处理(online analytical processing, OLAP)，但不适用于联机事务处理(online transaction processing, OLTP)

- 元组重构代价高
- 元组删除和修改代价高
- 解压代价

Example (元组重构)

CS-001	Elsa	F	19	CS
CS-002	Ed	M	19	CS
MA-001	Abby	F	18	Math
PH-001	Nick	M	20	Physics

```
SELECT * FROM Student WHERE Sno = 'MA-001';
```

Summary

实验1: 存储管理

内容

- 磁盘管理器
- 记录管理器
- 缓冲区管理器
- 页面替换策略

时间

- 1星期