

第八章 关系数据库引擎基础

1 数据库存储

2 缓存

3 散列表

4 查询处理

1 数据库存储

1.1 数据存储概述

1.2 数据库存储结构

文件、页、元组、日志

1.3 系统目录

1.4 存储模型

NSM、DSM

1.1 数据存储概述

- 面向磁盘的存储架构
- 用于数据库的存储介质及其架构
- 磁盘性能的度量
- 磁盘块访问的优化
- 面向磁盘的DBMS VS. OS

1.1.1 面向磁盘的存储架构

我们已经从概念层、逻辑层的角度，认识到关系模型中的数据库是表的集合，并能使用SQL实现读/写操作。

本章开始，将了解DBMS如何管理DB，包括DBMS如何简化和协助对数据的访问，如何实现物理细节对用户透明。

问题背景：

- 磁盘、内存，易失/非易失的问题
- 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。

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

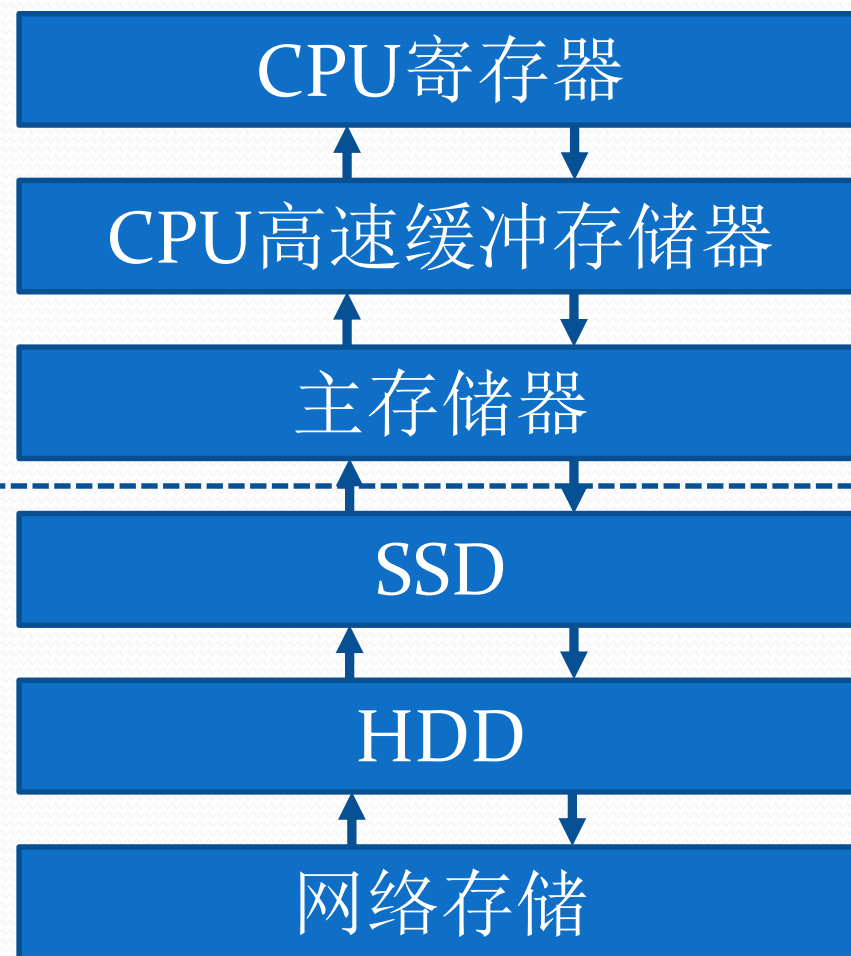
1.1.2 用于数据库的存储介质及其架构

用于数据库数据存储的主要介质：

- 高速缓冲存储器（一级、二级）
- 主存储器
- 快闪存储器
- 磁盘（HDD、SSD）
- 光盘
- 网络存储
- 磁带

1.1.2 用于数据库的存储介质及其架构

- 易失
- 随机访问
- 字节可寻址



- 非易失
- 顺序访问
- “块”可寻址

最新研究：非易失性内存

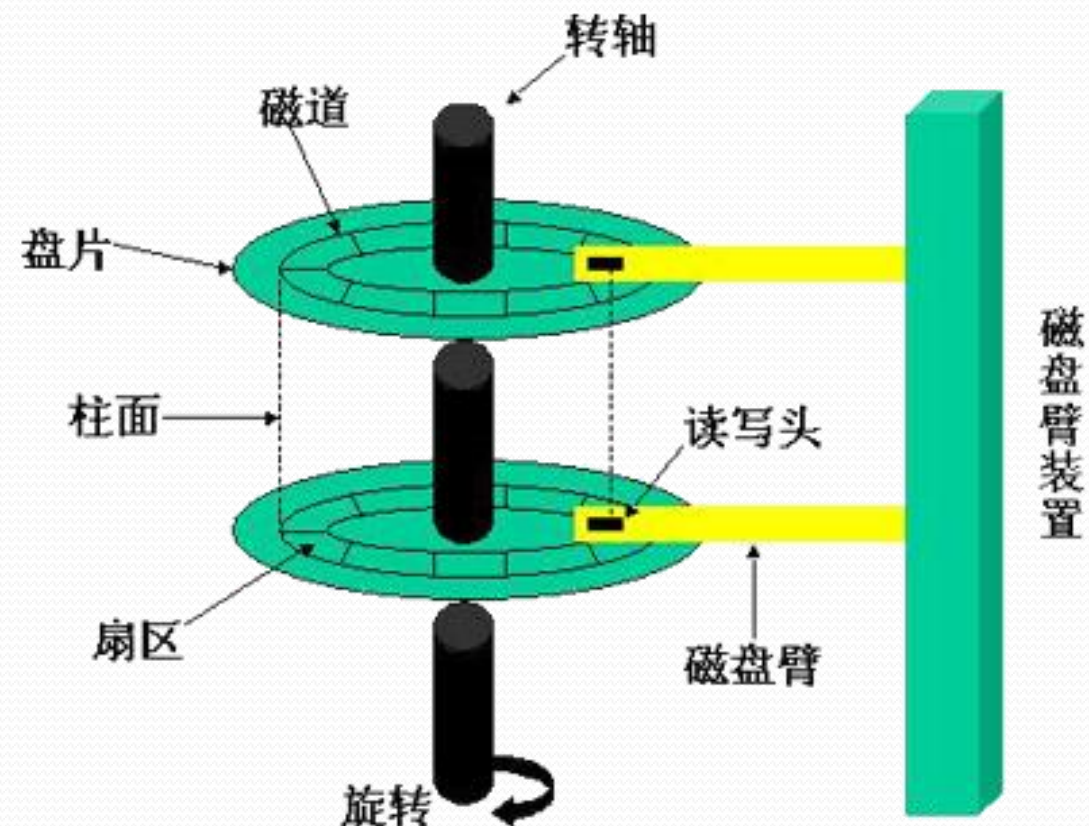
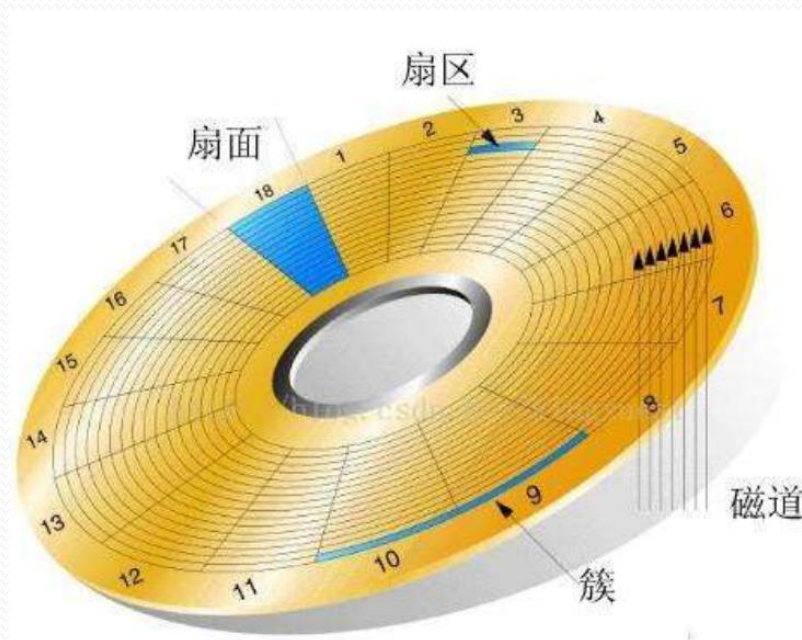
访问速度快
容量小
昂贵

访问速度慢
容量大
便宜

1.1.2 用于数据库的存储介质及其架构

磁盘的物理术语:

- 磁盘臂, 读写头, 转轴, 盘片, 柱面, 磁道 (约50000-100000条, 500-2000个扇区), 扇区 (读、写最小单位, 512Bytes)



磁盘性能的度量（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

磁盘性能的度量

磁盘性能的主要度量指标:

- 容量
- 访问时间（发出请求——数据开始传输，访问时间 = 寻道时间 + 旋转等待时间，寻道时间通常占一半）
- 数据传输率（读、写数据的速率，磁盘外侧比内侧快约2-3倍）
- 可靠性（常用标准是“平均故障时间”）

1.1.3 磁盘块访问的优化

文件系统、虚拟内存管理器发出“磁盘I/O请求”，每个请求指定了需要访问的磁盘地址（“块”号），磁盘和主存储器之间数据传输的单位为“块”。

访问请求可分为顺序和随机访问模式

➤ 顺序访问 (Sequential Access)

- 连续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
- DBMS会尽可能多的选择顺序访问而非随机访问

➤ 随机访问 (Random Access)

- 每一次请求都需要“磁盘寻道”，其效率低于顺序访问模式

1.1.3 磁盘块访问的优化

I/O操作代价较高，DBMS领域为提高访问块的速度，形成了很多技术：

➤ 缓冲（Buffering）

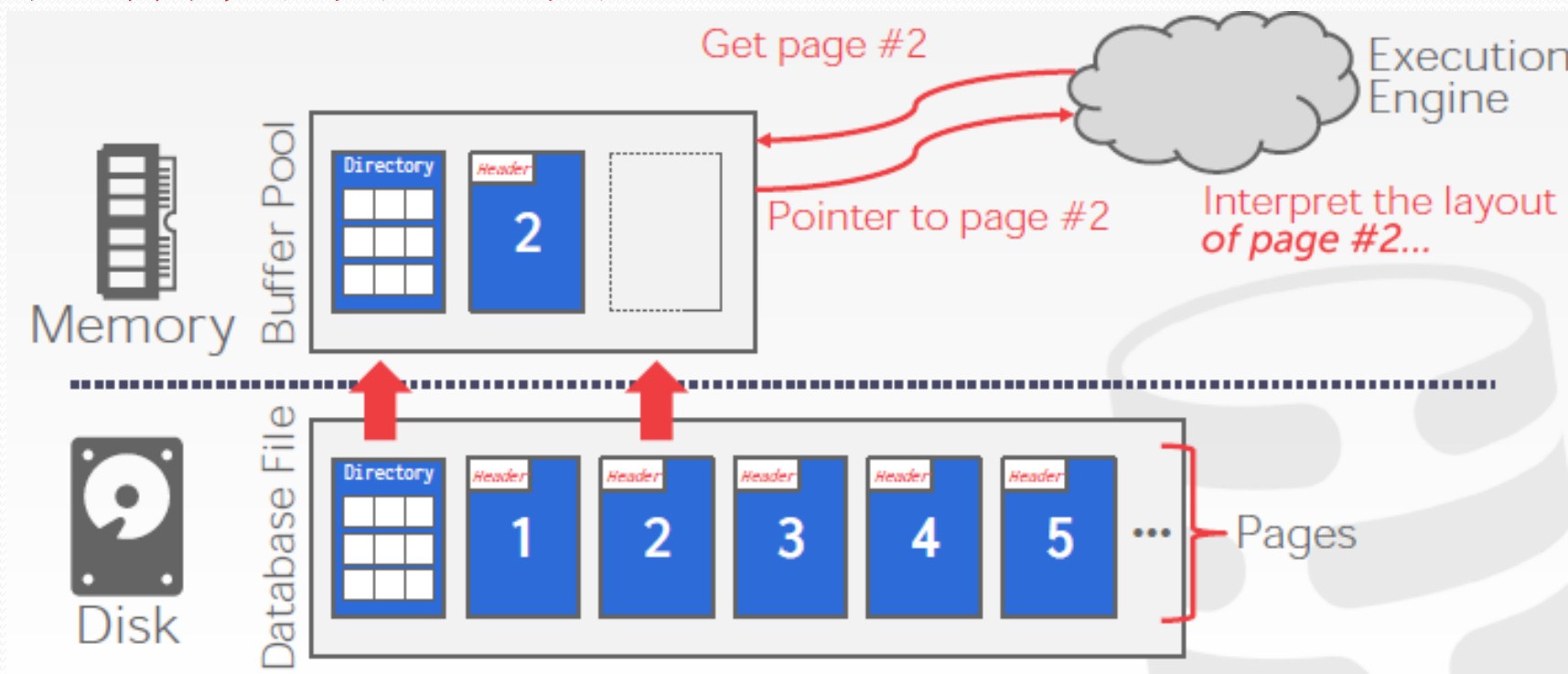
- 从磁盘读取的块暂时存储在内存缓冲区中，以满足将来的需求。
- 缓冲区通过操作系统和数据库系统共同运作。

➤ 预读（Read Ahead）

数据预取技术，即在读取指定数据的同时也预先读取与其相邻的一定范围内的数据。

1.1.4 面向磁盘的DBMS VS. OS

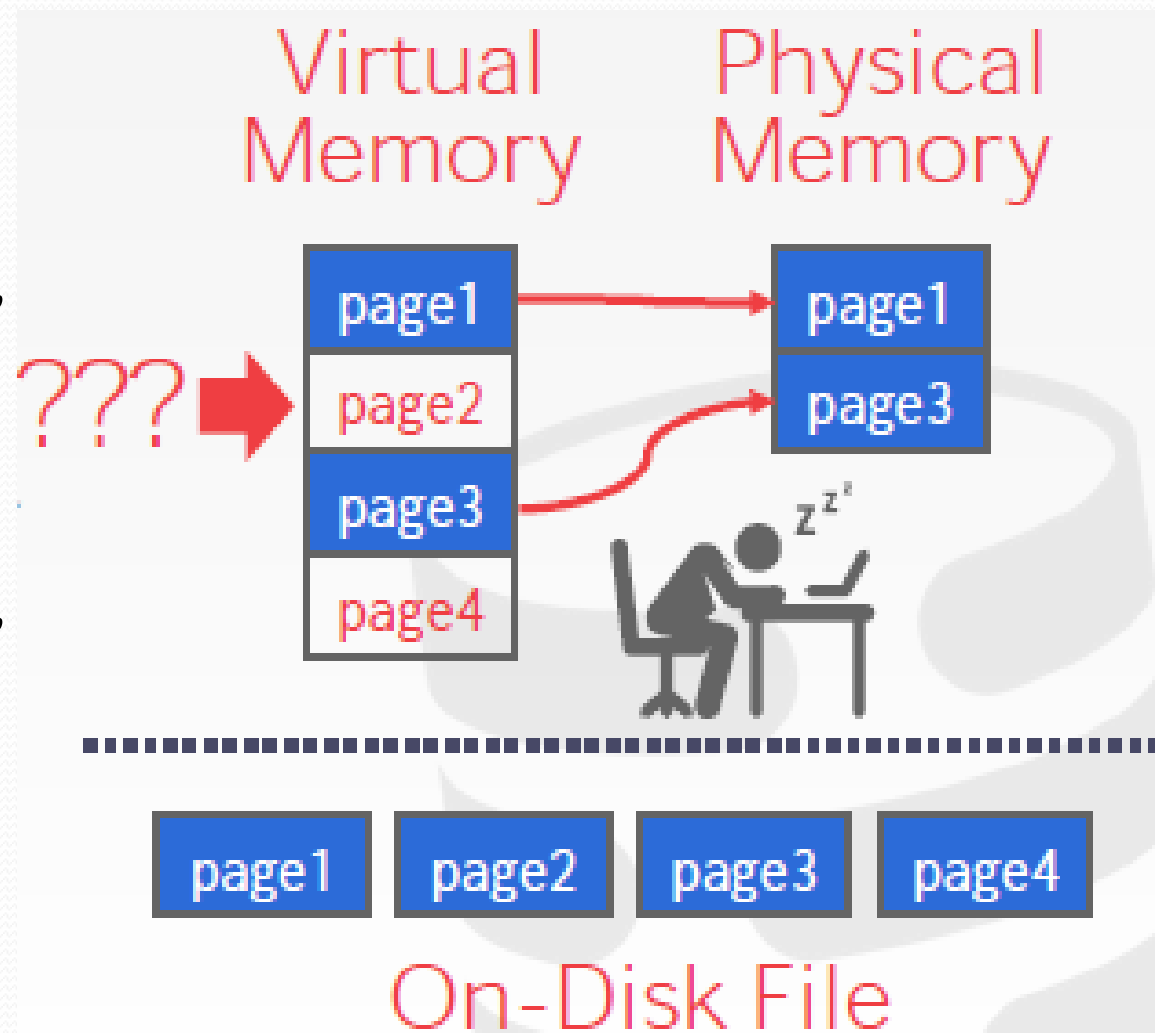
例：当需要访问“page2”数据时，执行引擎首先查看缓冲池中是否存在page2，如果没有则调用存储管理从磁盘读取并缓存page2。缓冲池需要保障不发生“缺页”！



1.1.4 面向磁盘的DBMS VS. OS

上例中，能否使用操作系统替代DBMS？

- 一种常规思路是MMAP (memory mapping)，MMAP将数据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。
- 如果是“读”操作，当出现“缺页”时，进程被“阻塞”，OS将page从磁盘load到内存。



1.1.4 面向磁盘的DBMS VS. OS

如果是“写”操作，由于日志、并发控制等实现的需要，操作系统并不知道哪些page需要在其他page之前写到磁盘

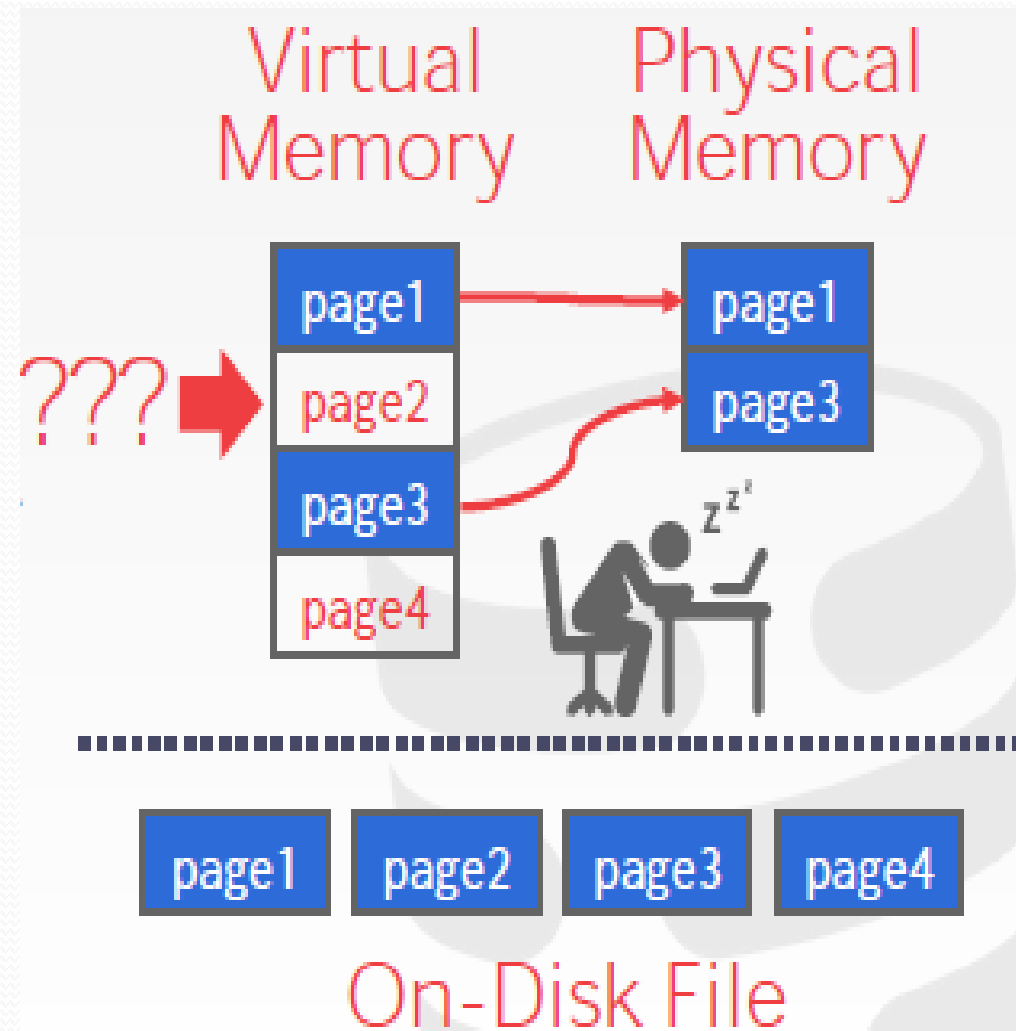
- 可能的解决方法是引导OS的页缓存替换机制：

 madvise: 告诉OS期待读取特定页或内存使用模式；

 mlock: 告知OS某些内存范围不能被替换出去；

 msync: 告知OS某些内存范围被写出到磁盘。

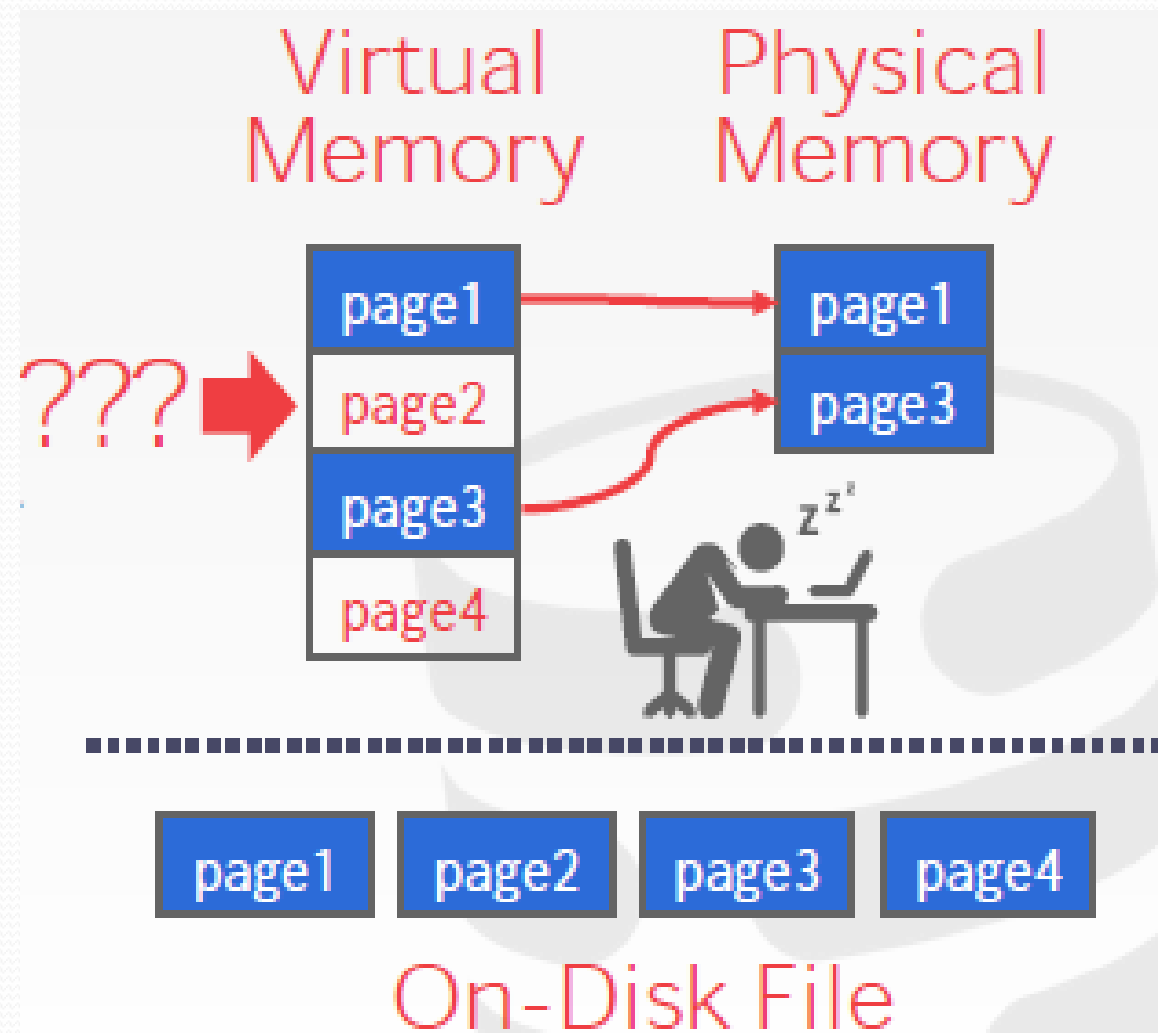
早期的MongoDB如此，后来它收购wiredtiger作为其默认的存储引擎。



1.1.4 面向磁盘的DBMS VS. OS

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。



1.2 数据库存储结构

1.2.1 文件组织

基于链表的堆文件，基于页目录的堆文件

1.2.2 页设计

面向元组的页，槽页

1.2.3 元组设计

1.2.4 日志设计

1.2.1 文件组织

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)

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）。

数据库页面

- 页面是一个固定大小的数据块block。
 - 它可以包含元组、元数据、索引、日志记录...
 - 大多数系统不会混合页面类型。

页的堆文件组织方式

不同的DBMS管理磁盘中pages的方式不同，堆文件组织（Heap File Organization）是一种常见的方式。

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pages的操作

堆文件的两种表示方式

- 链表（Linked List）
- 页目录（Page directory）

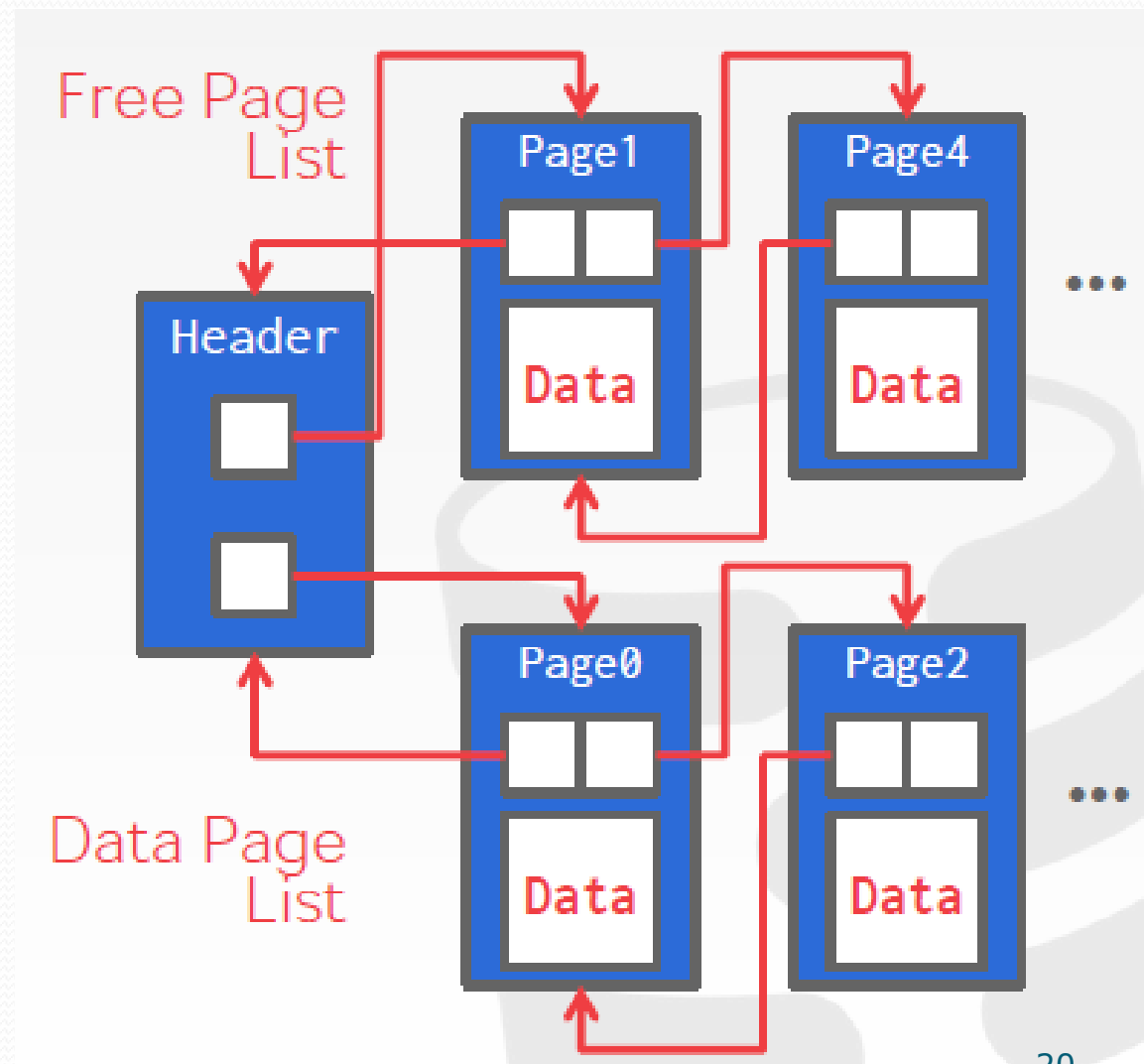
多文件时，需要元数据记录文件中有哪些页面，以及哪些页有空闲空间。

页的堆文件组织：链表

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 list）头部
- 数据页列表（data list）头部

每个page均记录当前空闲的空间
(slot)

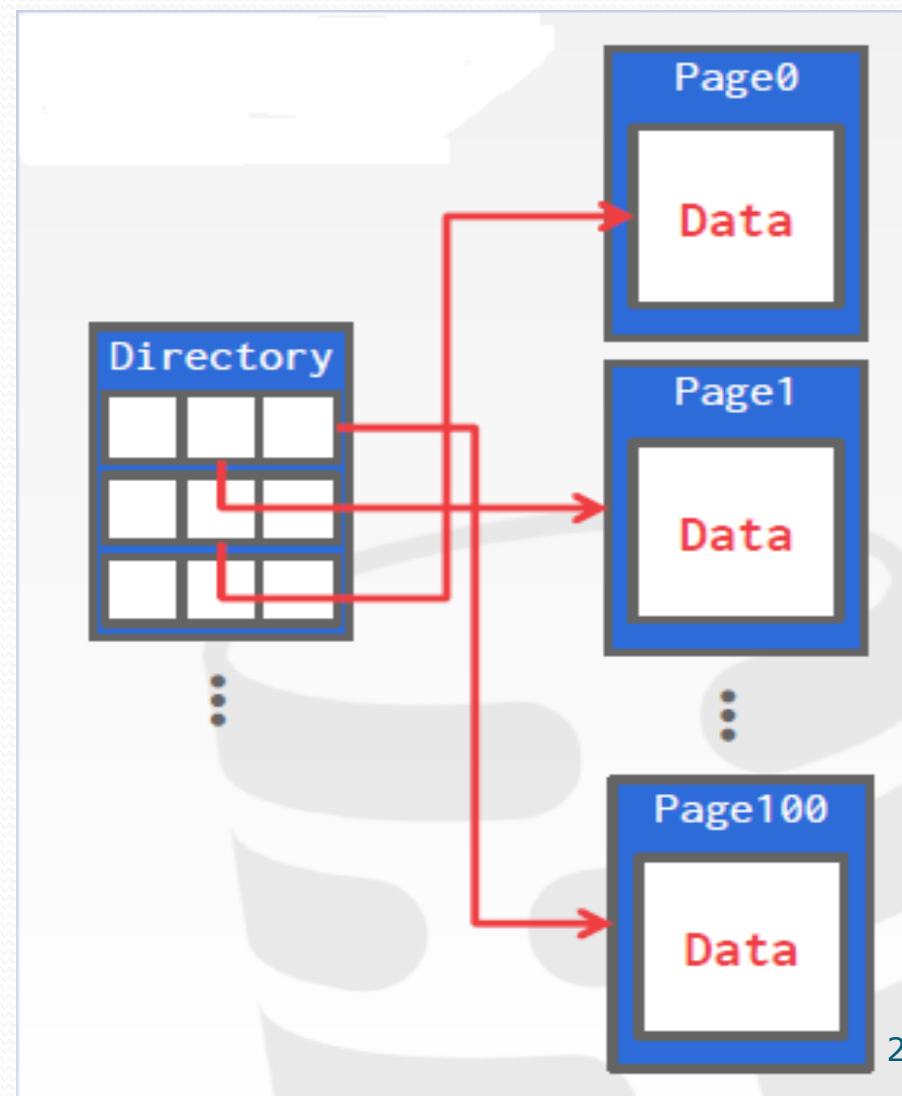


页的堆文件组织：页目录

堆文件中设立一类专门的页面（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闲空间信息（slot）。

DBMS必须保持目录页与所有页的当前信息同步。



1.2.2 页设计 (Page Layout)

数据库的页具备固定大小，页可以容纳：

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）

页的大小 (Size of Page)

硬件页面 (4KB)

操作系统页面 (4KB)

数据库页面 (512B-16KB)

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。

4KB



ORACLE

8KB



16KB



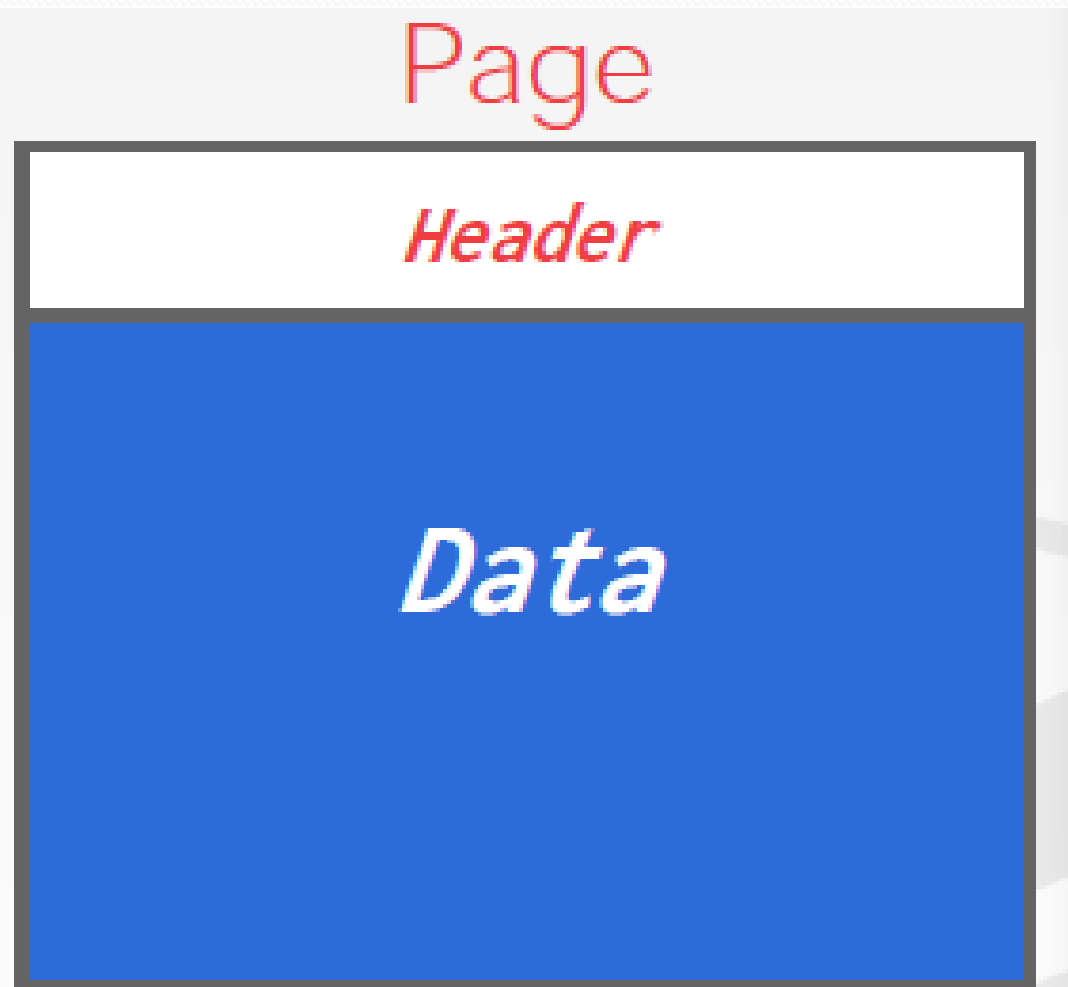
页头 (Page Header)

每个页面都有页头 (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

页面内“数据”的组织方式：

- 面向元组型
- 日志结构型



面向元组型的页设计

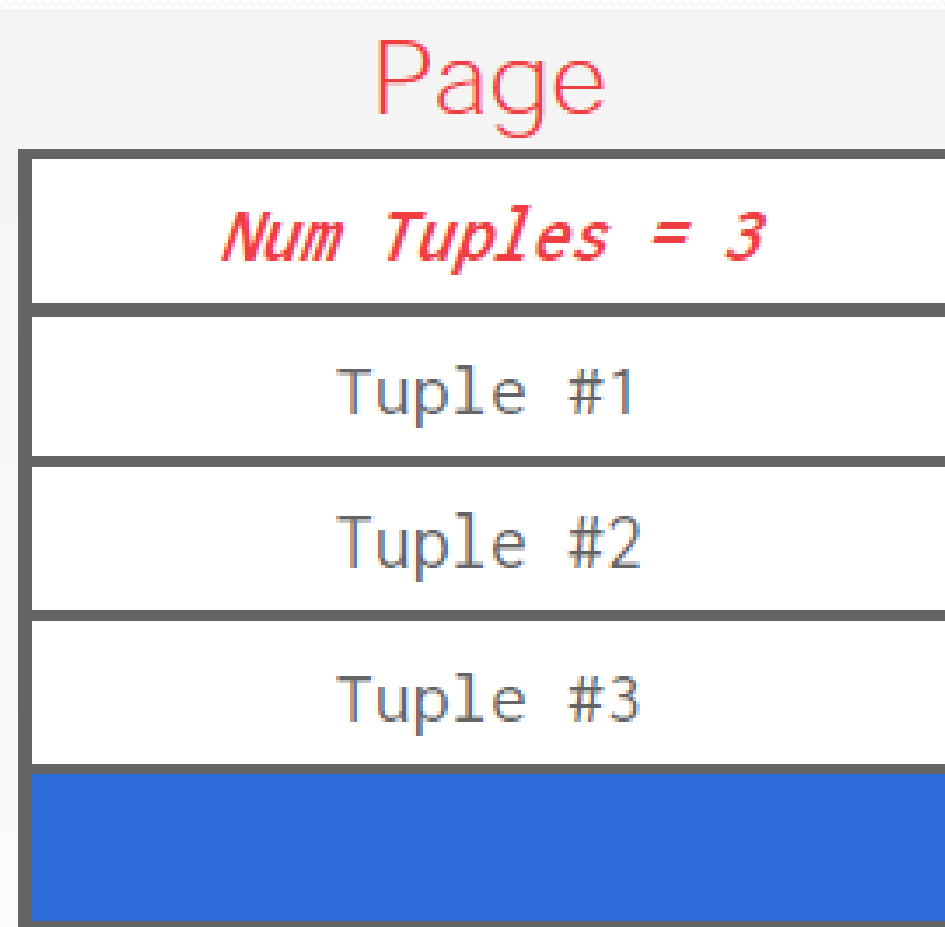
基本思想:

- header记录页内的元组数，类似数组的方式进行存储；
- 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
- 变长的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)

思路来自slot数组

Slot数组将“槽位”映射到特定元组
开始位置的偏移量

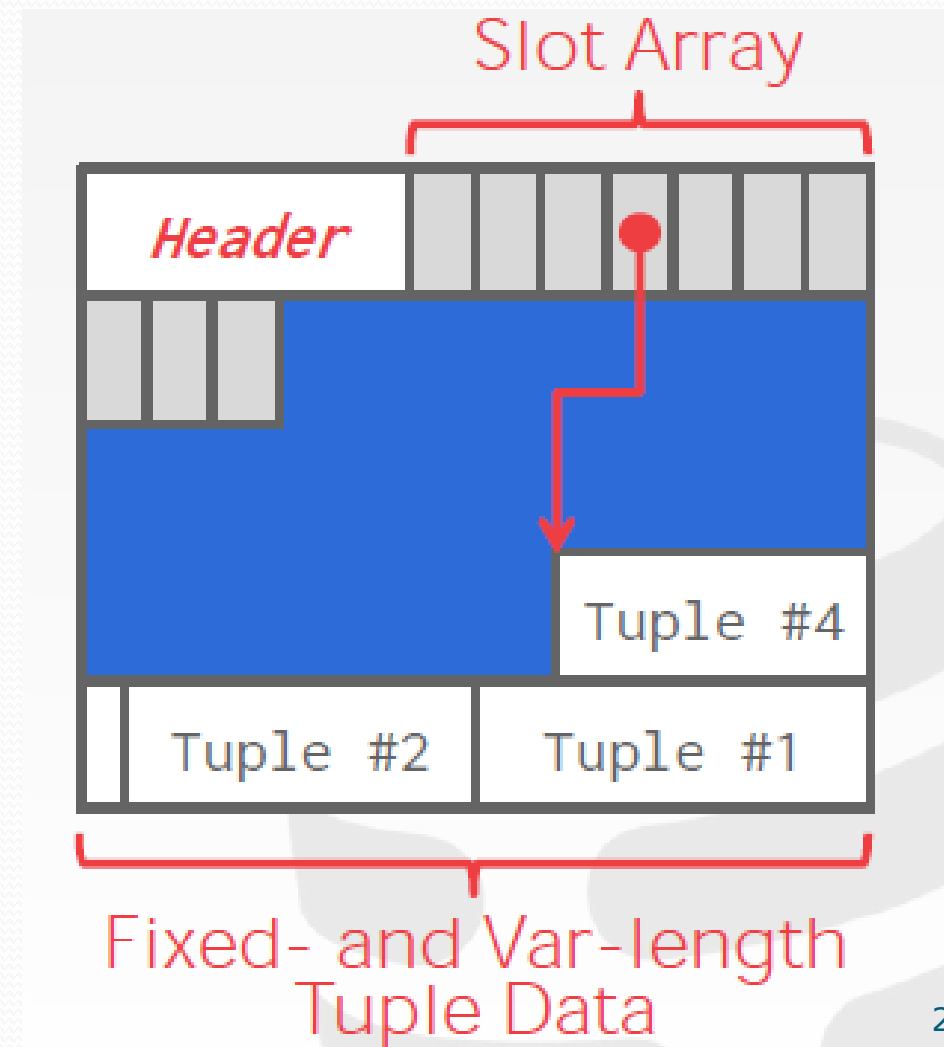
Header记录:

- 已占用的槽位;
- 以及上一次使用槽位的开始位置。

元组: 在页内倒序存放。

元组在内部的唯一标识符: 可以使用page id和slot id (或偏移量), 也可包含文件位置信息。

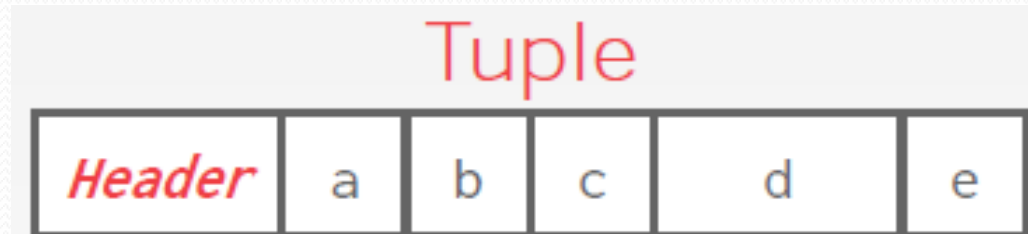
定长、变长元组轻松应对



1.2.3 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。

每个元组有一个前缀为header包含元数据



```
CREATE TABLE foo (  
  a INT PRIMARY KEY,  
  b INT NOT NULL,  
  c INT,  
  d DOUBLE,  
  e FLOAT  
);
```

1.2.3 元组设计 (Tuple Layout)

物理上非规范化 (Denormalize) : (“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),  
);
```

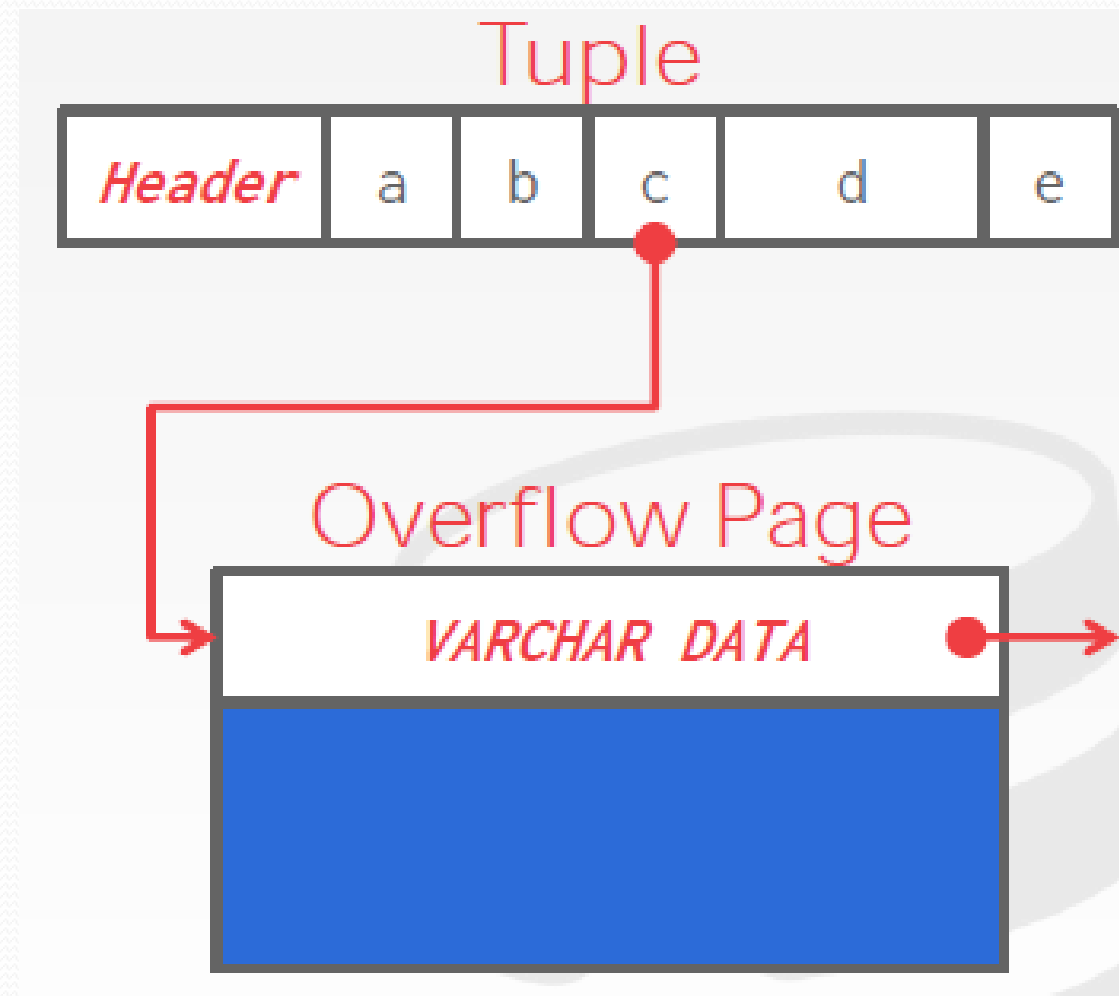
“大”值 (Large Values) 存储

多数DBMS不允许一个元组的大小超过页的大小

为了存储超过一页的“大数据”，一些DBMS使用“溢出存储页 (overflow)”

- Postgres: TOAST (>2kb)
- Mysql: Overflow (>1/2 size of page)
- SQL Server: Overflow (> size of page)

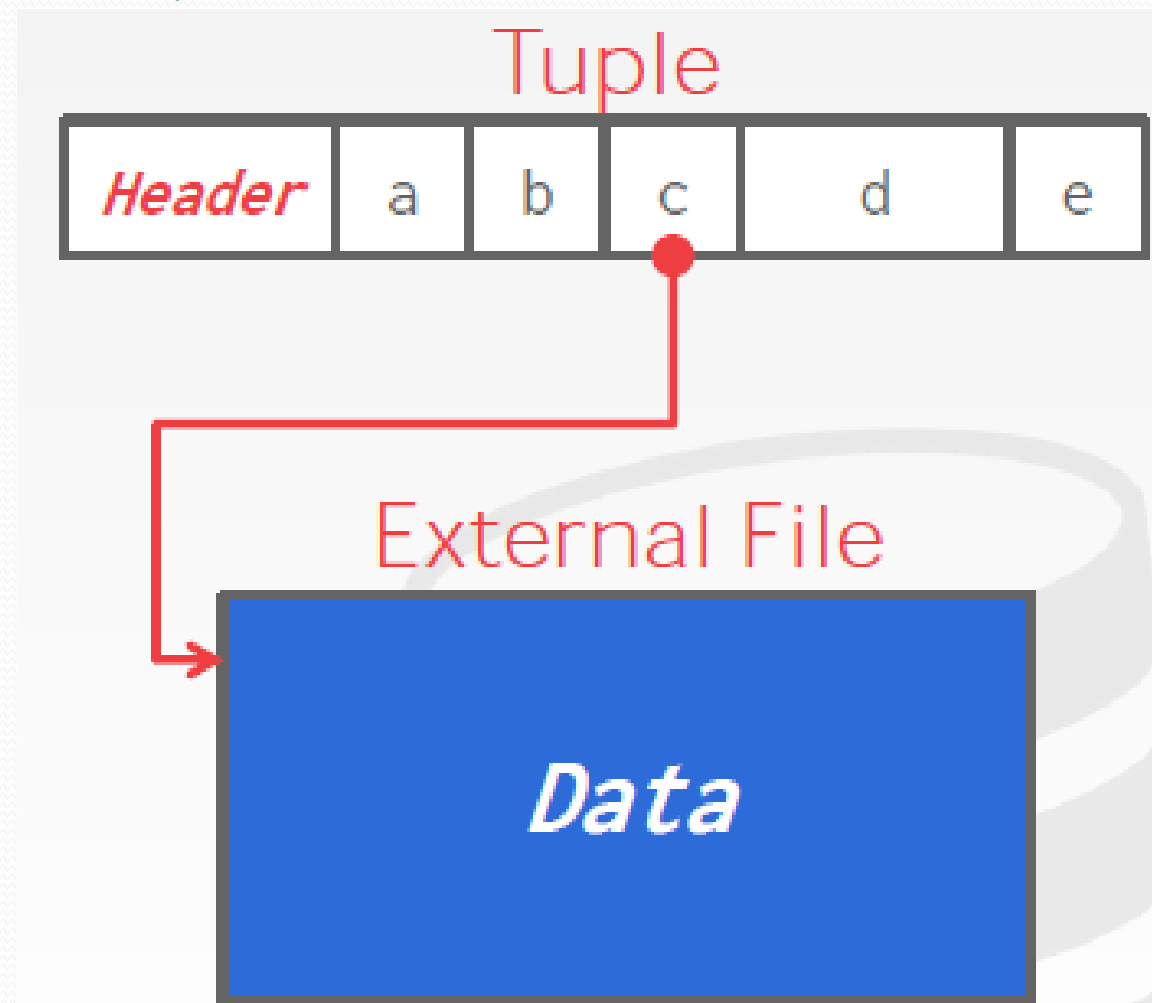
超过页大小时，另一种情况是“BLOB”类型。



“大”值（Large Values）存储

有些DBMS允许将一个大值存放在外部文件中，作为一个“BLOB”类型。比如Oracle的BFILE类型和SQL Server的FILESTREAM类型

显然，此时DBMS无法操作外部文件的内容，没有持久性、事务保障，仅仅是管理。



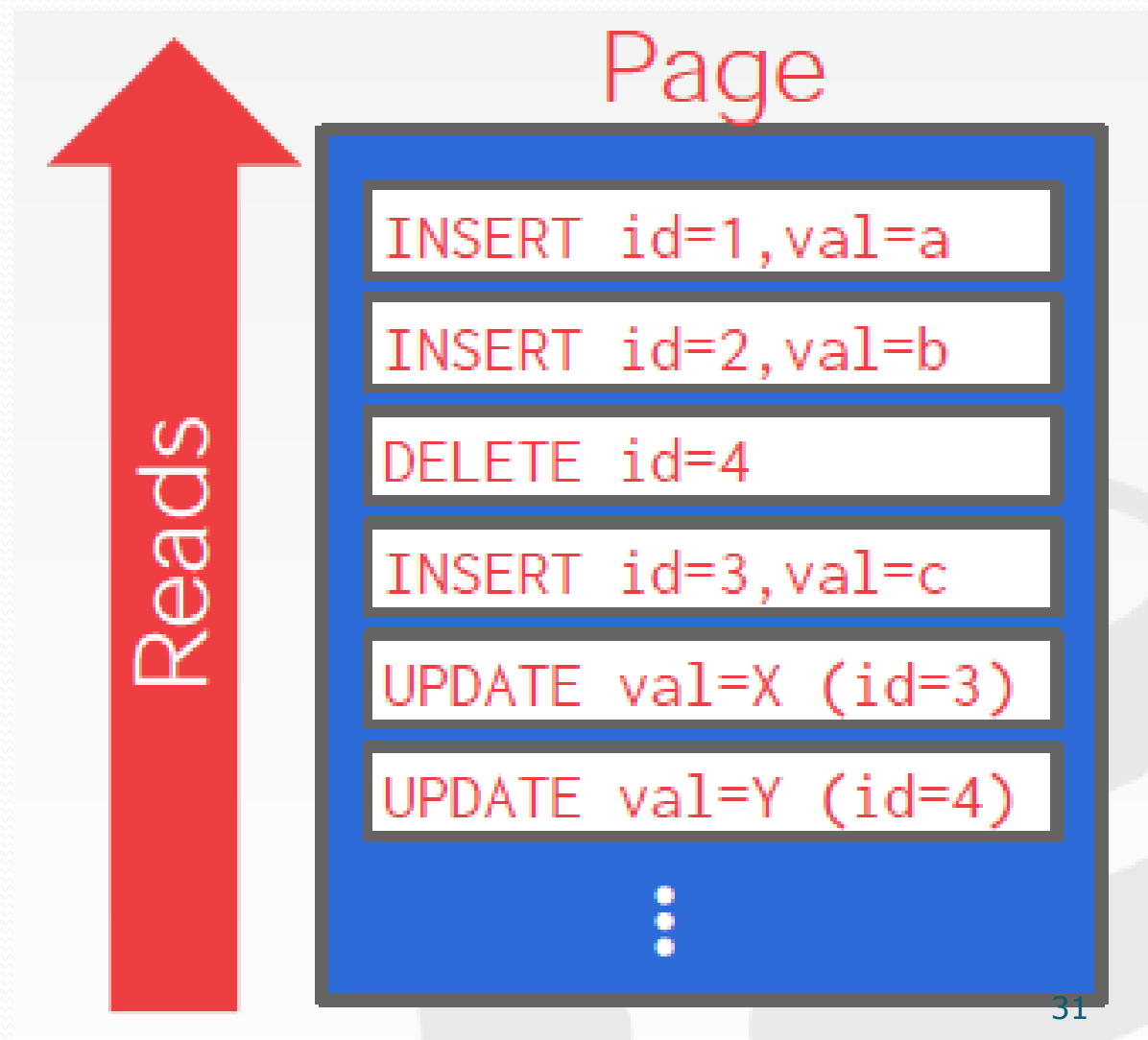
1.2.4 日志式文件组织

页中只存放日志记录

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，溯源找到某数据源头，还可以“回滚”。



1.2.4 日志文件组织

优化方法:

- ✓ 可建立“日志索引”，方便查找相关的日志记录
- ✓ 日志可定期压缩（通过删除不必要的记录来合并日志文件）。

Page

```
id=1,val=a  
id=2,val=b  
id=3,val=X  
id=4,val=Y
```

1.3 系统目录 (System Catalogs)

DBMS将数据库的元数据（描述信息）存放在内部的目录（数据字典）中：

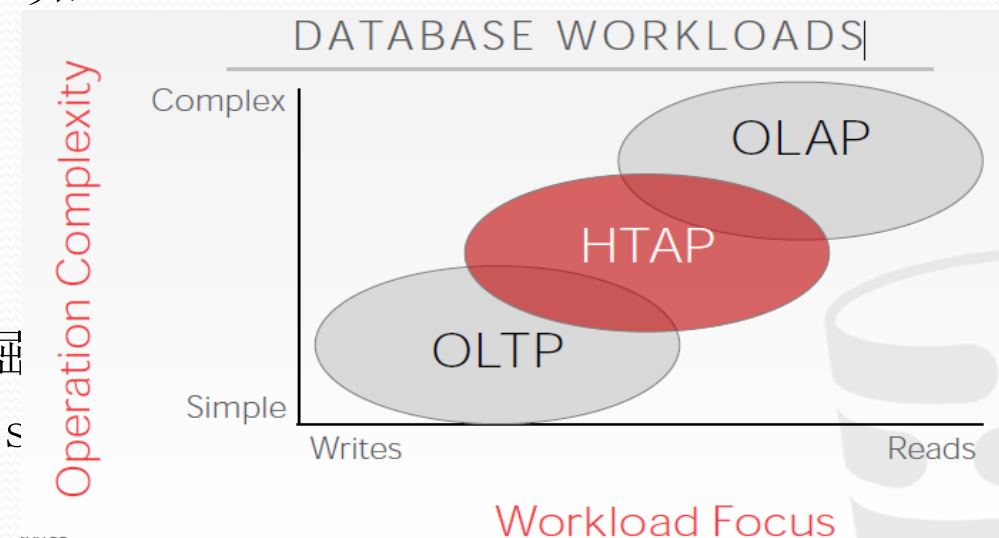
- 表、列、索引、视图
- 用户、权限
- 数据的统计信息
- 存储过程、触发器等

很多DBMS将系统目录保存在一个数据库中，例如SQL Server的master数据库。

1.4 存储模型 (Storage Model)

数据库的存储模型从全局、应用特征等角度，尤其大数据环境，考虑数据库如何适应需求。这里考虑的“**工作量 (Workloads)**”如下：

- 联机事务处理 (On-Line Transaction Processing, OLTP)
 - 传统具较强“事务特性”需求的应用，比如电商、贸易等
- 联机分析处理 (On-Line Analytical Processing, OLAP)
 - 数据量较大，主要是查询、复杂查询、统计，甚至数据挖掘
- 复合事务分析处理 (Hybrid Transaction-Analytical Process HTAP)
 - 兼具OLTP和OLAP特征



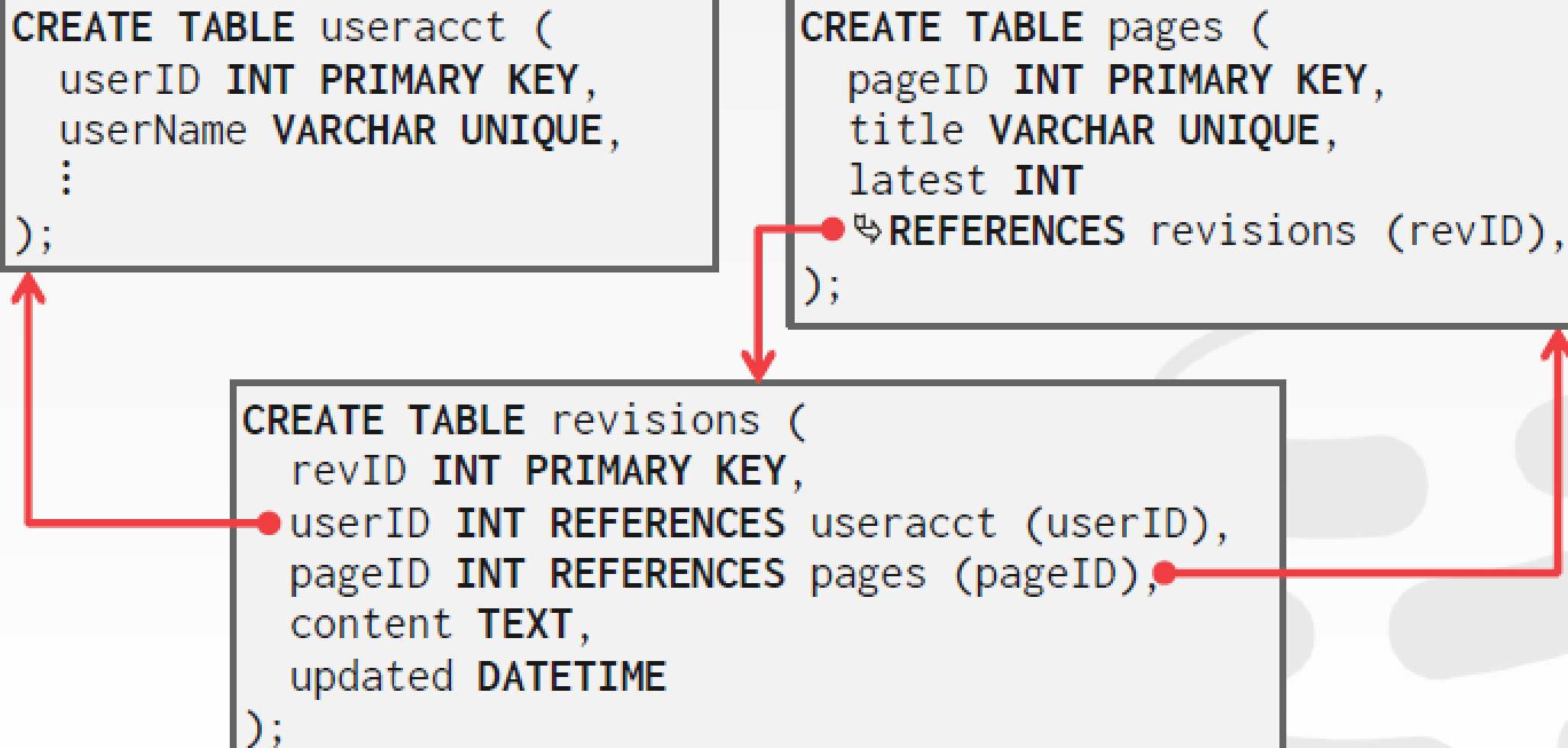
1.4 存储模型

维基百科例子

```
CREATE TABLE useracct (  
  userID INT PRIMARY KEY,  
  userName VARCHAR UNIQUE,  
  :  
);
```

```
CREATE TABLE pages (  
  pageID INT PRIMARY KEY,  
  title VARCHAR UNIQUE,  
  latest INT  
  REFERENCES revisions (revID),  
);
```

```
CREATE TABLE revisions (  
  revID INT PRIMARY KEY,  
  userID INT REFERENCES useracct (userID),  
  pageID INT REFERENCES pages (pageID),  
  content TEXT,  
  updated DATETIME  
);
```



1.4 存储模型

当应用在OLTP中，我们通常运行一些较为简单的“读/写”SQL语句，以实现我们的业务计算

而在OLAP应用中，查询语句往往非常复杂，甚至可能需要用到多个不同数据库。因此有时候不得不收集数据后，将这些工作负载交给服务器来处理

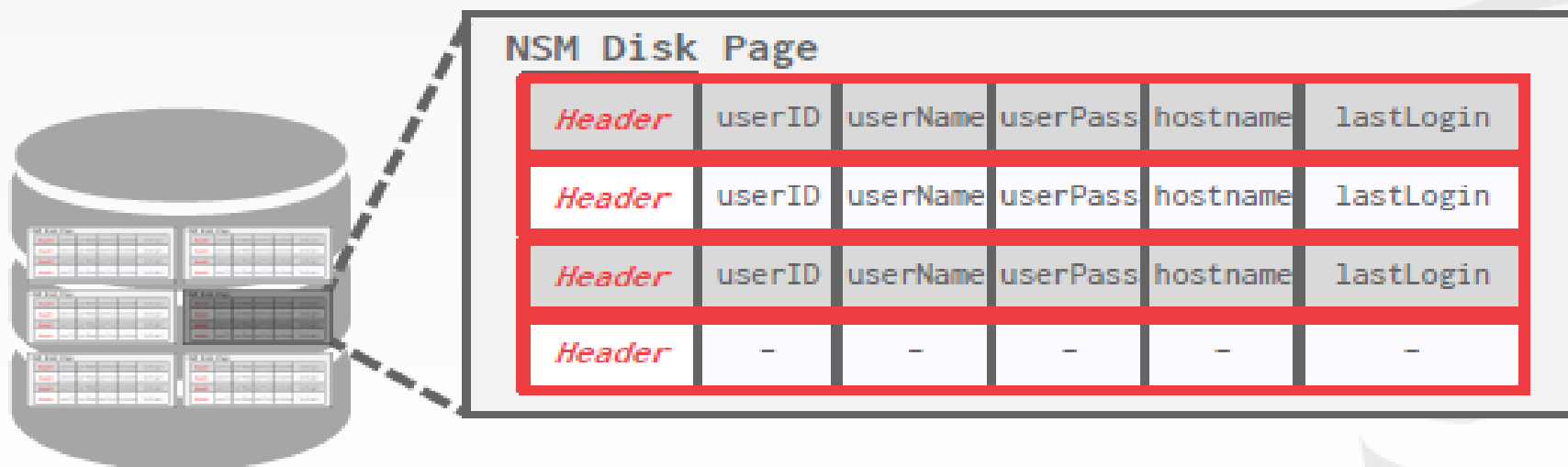
```
SELECT COUNT(U.lastLogin),  
       EXTRACT(month FROM  
               U.lastLogin) AS month  
FROM useracct AS U  
WHERE U.hostname LIKE '%.gov'  
GROUP BY  
       EXTRACT(month FROM U.lastLogin)
```


1.4 存储模型

NSM

为适应OLTP或OLAP不同的工作负载，DBMS可以采用不同的方式进行元组的存储。

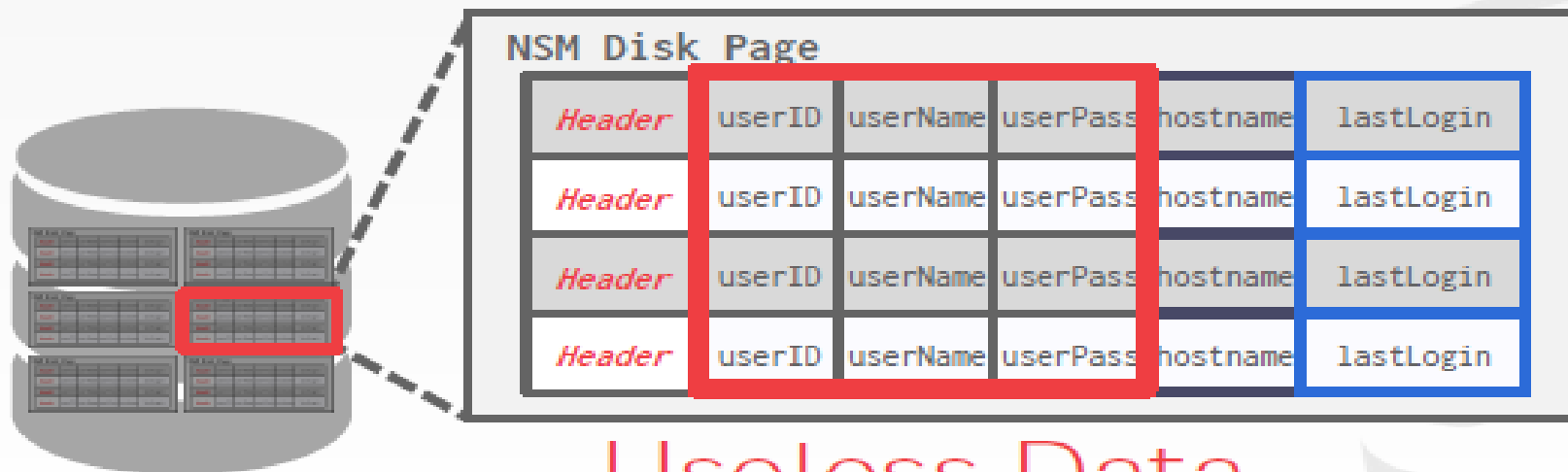
常见的n元存储模型（n-ary storage mode, NSM，又名“行存储”）非常适合OLTP。此时，单个元组的所有属性连续的分布在一个page中，查询往往涉及单个实体（工作量较少），并能适应较为繁重的“更新”工作量



1.4 存储模型

OLTP VS OLAP, 不同的数据处理需求

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
FROM useracct AS U
WHERE U.hostname LIKE '%.gov'
GROUP BY EXTRACT(month FROM U.lastLogin)
```



1.4 存储模型

从上例可以看出NSM优缺点：

- 优点：适合OLTP，对输出结果是全部属性的查询，对快速的增、删、改操作非常友好；
- 缺点：不适合查询table的部分属性。

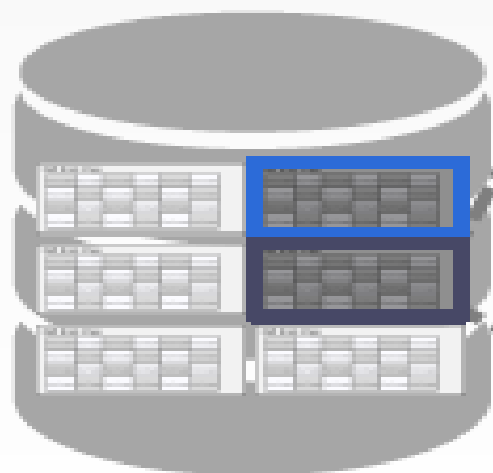
DSM

针对OLAP，分解存储模型（Decomposition Storage Model, DSM）更为适合，又称为“列存储”，DBMS将单个属性的值连续的组织在一个page中；

- 可以很好的适应大数据量、复杂查询语义、高负载查询。

DSM存储模型

```
SELECT COUNT(U.lastLogin),  
       EXTRACT(month FROM U.lastLogin) AS month  
FROM useracct AS U  
WHERE U.hostname LIKE '%.gov'  
GROUP BY EXTRACT(month FROM U.lastLogin)
```



DSM Disk Page

hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname

DSM存储模型

DSM优点

- 由于只读取需要的数据，因此减少了I/O次数；
- 更便捷的查询处理；
- 有利于数据压缩的实现。

DSM缺点

- 元组被“拆分”，有些查询需要进行“缝合”，影响查询速度，也同时影响增删改效率

1 数据库存储小结

- 存储管理器和DBMS的其他部分不是独立的
- 结合目标负载类型选择合适的存储模型很重要