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3章 作业问题

(Chap3-4) 根据第二章习题6中的4个表（SPJ）建表，并完成第二章习题6中的查询。

正确答案：

建表略，注意要在建表语句中给出完整性定义。

(4) 求没有使用天津供应商生产的红色零件的工程号JNO。

写法1：

```
SELECT JNO FROM J
    WHERE NOT EXISTS
    ( SELECT S.*,SPJ.*,P.* FROM S, SPJ, P
        WHERE SPJ.JNO=J.JNO AND SPJ.SNO=S.SNO AND
            SPJ.PNO=P.PNO AND S.CITY= ‘天津’ AND
            P.COLOR= ‘红’ );
//SQL写法与第二章中2-6（4）中关系代数表达式的一致。
```

写法2：

```
SELECT JNO FROM J
    WHERE JNO NOT IN
    ( SELECT JNO FROM S, SPJ, P
        WHERE S.SNO=SPJ.SNO AND
            SPJ.PNO=P.PNO AND
            S.CITY=‘天津’ AND P.COLOR=‘红’);
```

第3章 作业问题

(5) 求至少用了供应商S1所供应的全部零件的工程号JNO。

写法1：

```
SELECT DISTINCT JNO FROM SPJ  X
WHERE NOT EXISTS
( SELECT * FROM SPJ  Y
      WHERE SPJY.SNO='S1' AND
       NOT EXISTS
       ( SELECT * FROM SPJ  Z
          WHERE  Z.JNO = X.JNO AND
           Z.PNO = Y.PNO AND  Z.SNO = Y.SNO));
```

查询结果：{ }

写法2：

```
SELECT DISTINCT JNO FROM SPJ  X
WHERE NOT EXISTS
 ( SELECT * FROM SPJ  Y
   WHERE SPJY.SNO='S1' AND
    NOT EXISTS
    ( SELECT * FROM SPJ  Z
      WHERE  Z.JNO = X.JNO AND
               Z.PNO = Y.PNO));
```

查询结果：{J4}

按本文题意，写法1是正确的。注意：本题中不同供应商可能生产同名零件。

第八章　关系数据库引擎基础

Principles of Database Systems

第八章 关系数据库引擎基础

8.1 数据库存储

8.2 缓存

8.3 索引

8.1.1 数据存储概述

8.1.2 数据库存储结构

文件、页、元组、日志

8.1.3 系统目录

8.1.4 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

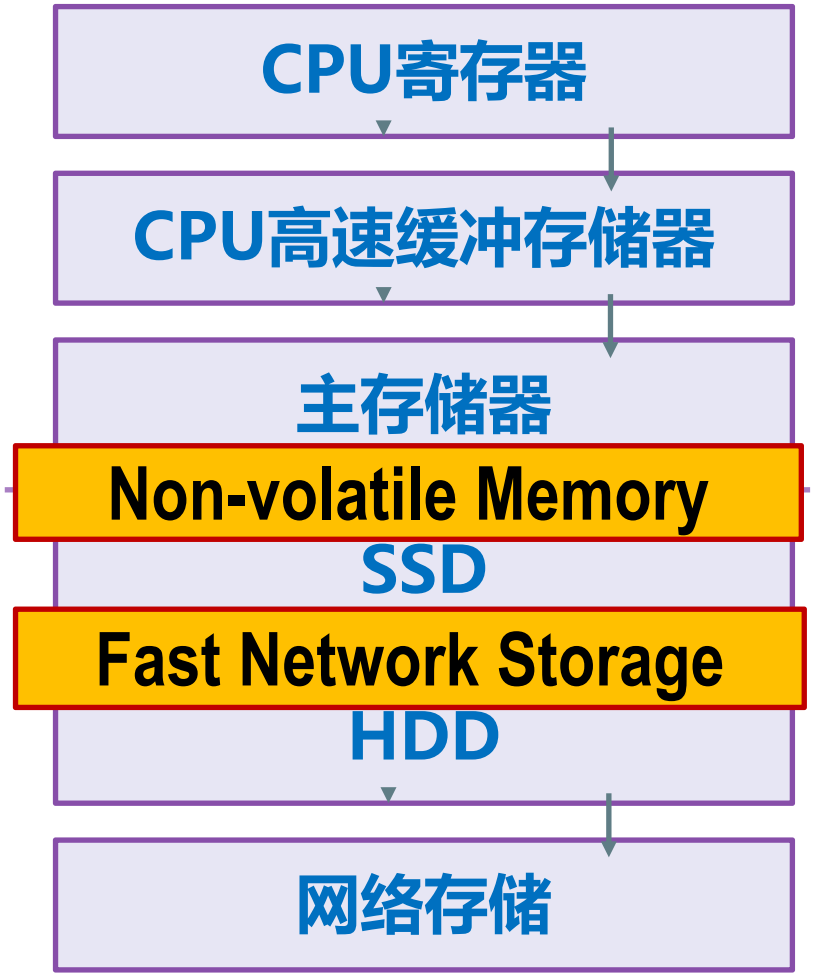
缓冲池管理器

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. 允许DBMS管理可超过内存大小的数据库。
2. 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。
3. 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。

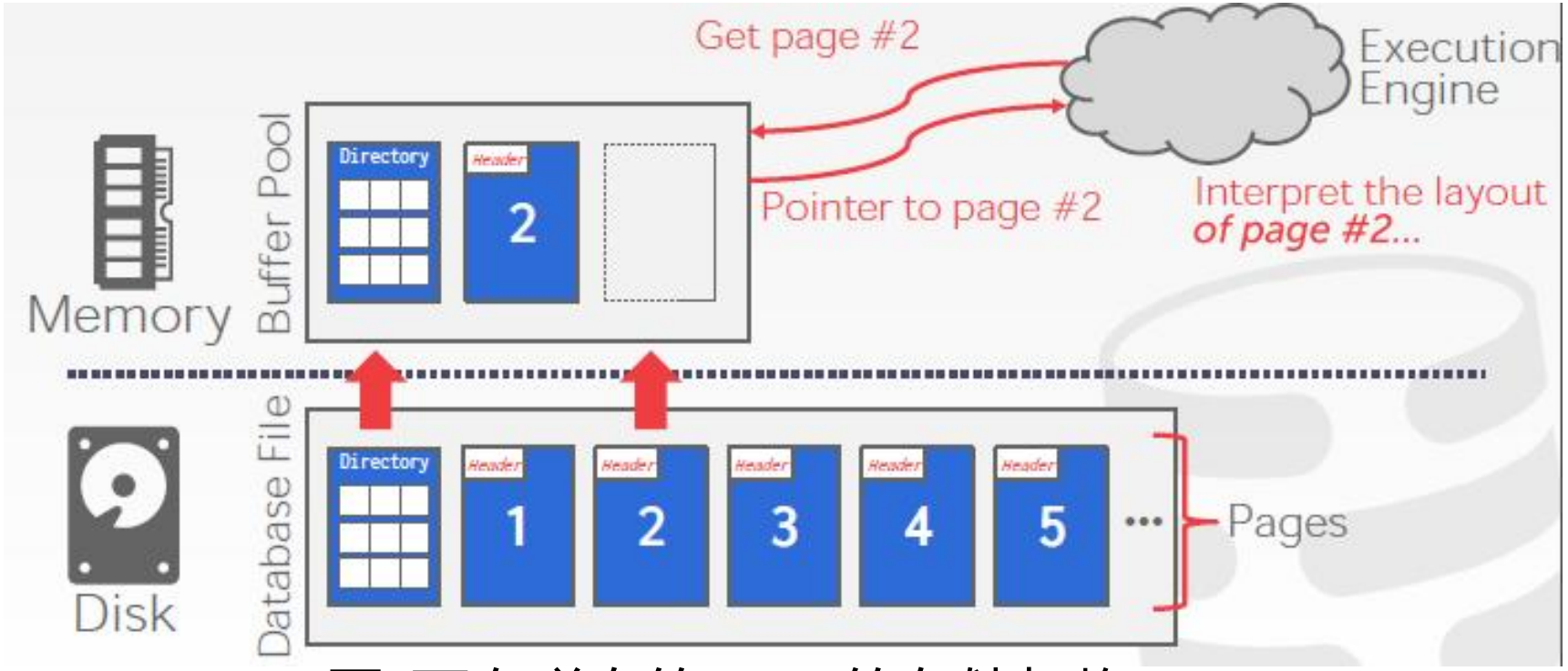
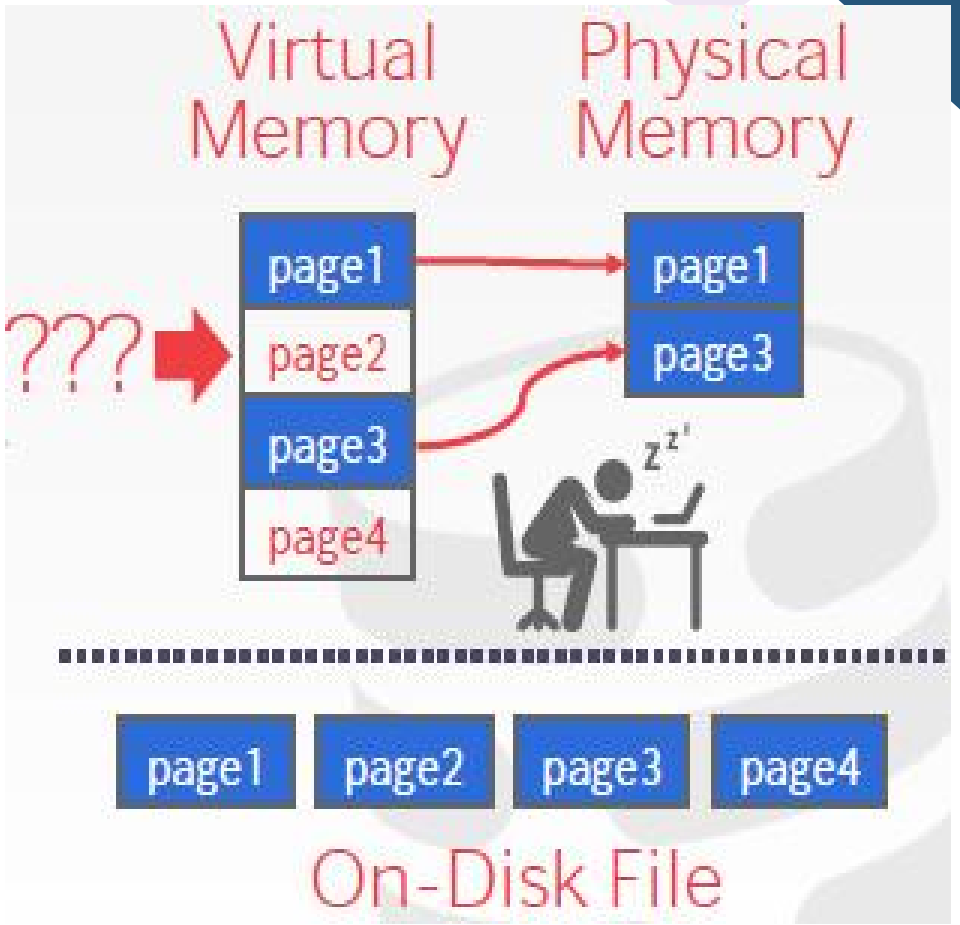


图 面向磁盘的DBMS的存储架构

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。

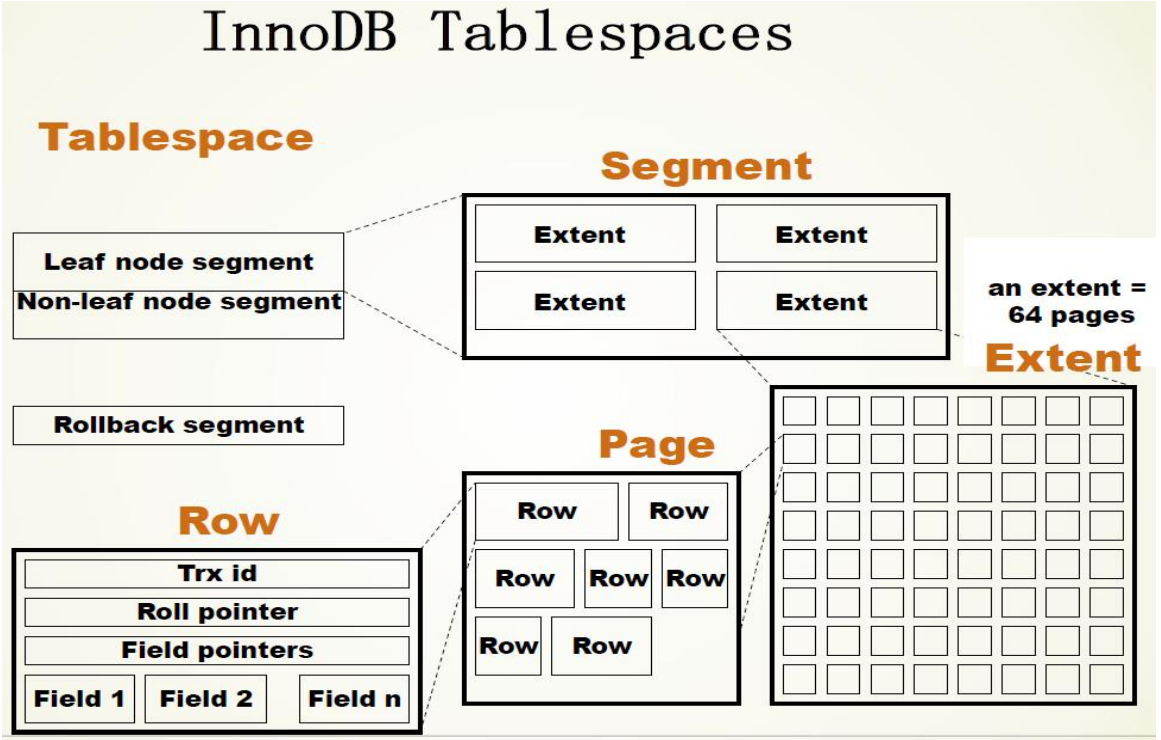
从效率、安全等角度出发，主流DBMS都倾向于自己来进行页面的管理，可更好的支持：

- **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

- 问题1：DBMS如何表示磁盘上存储的数据库文件？
  - 数据库存储结构
- 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 数据库存储结构

- 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 数据库存储结构

- 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。

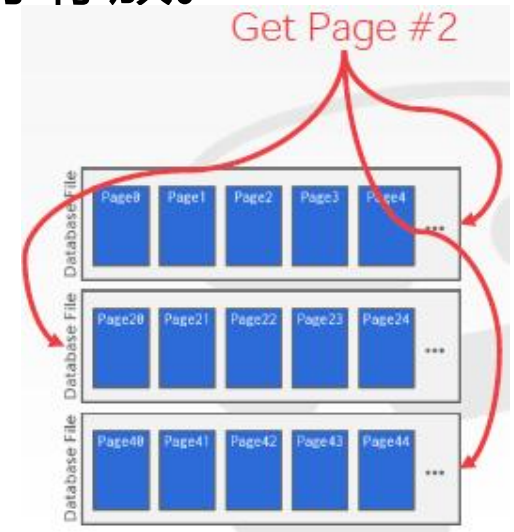
页的大小（Size of Page）

- 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

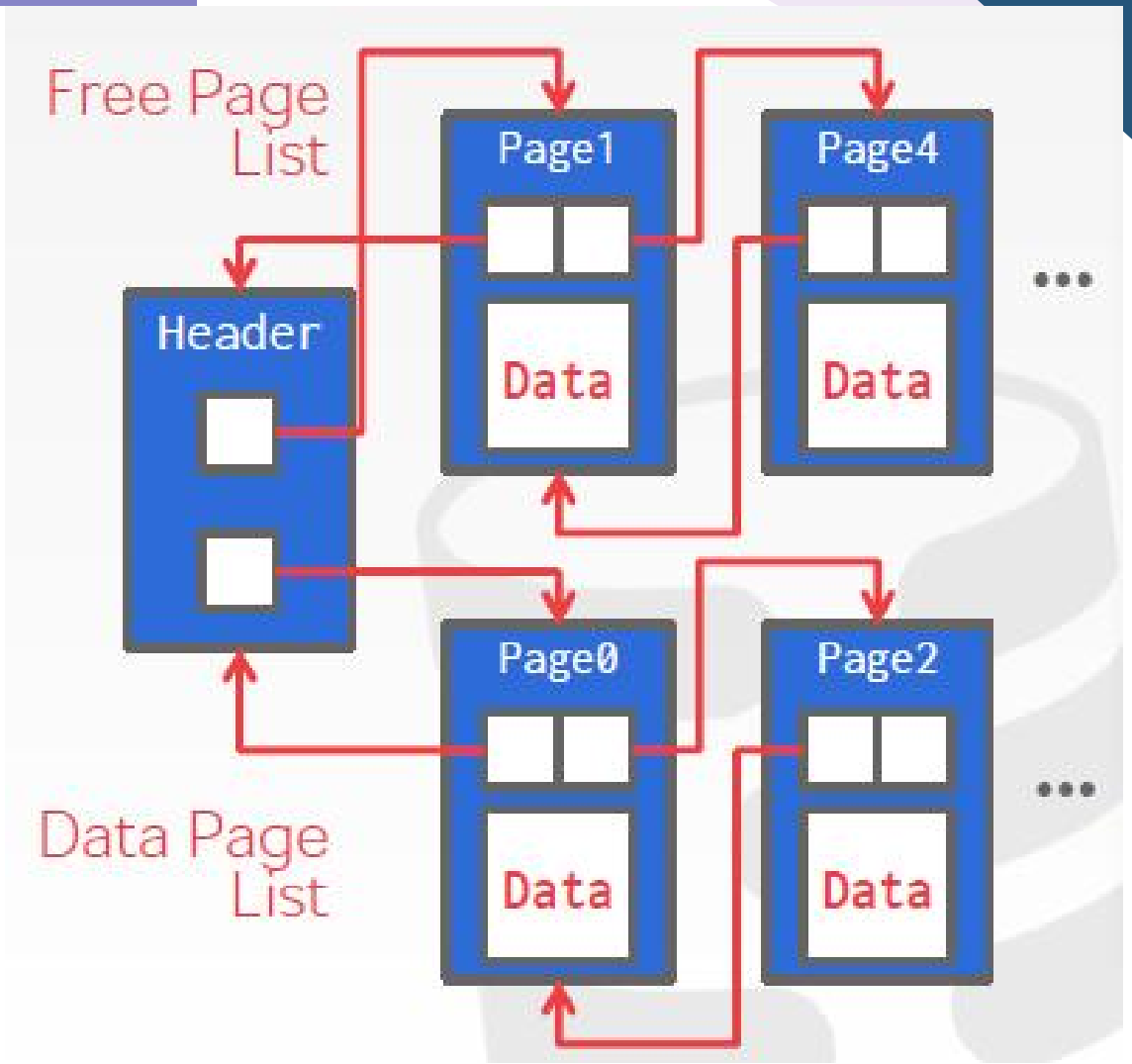
- 关系是记录的集合，这些记录在数据库文件中可以有3种组织方式：
  - 堆文件组织（Heap File Organization）
  - 顺序文件组织（Sequential File Organization）
  - 散列文件组织（Hash File Organization）
- Heap文件是一个无序的page集合，其中的元组可按随机顺序存放。
  - 支持page的创建、读、写和删除操作
  - 支持遍历所有pages的操作
- 堆文件的2种表示方式：
  - 链表（Linked List）
  - 页目录（Page directory）



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

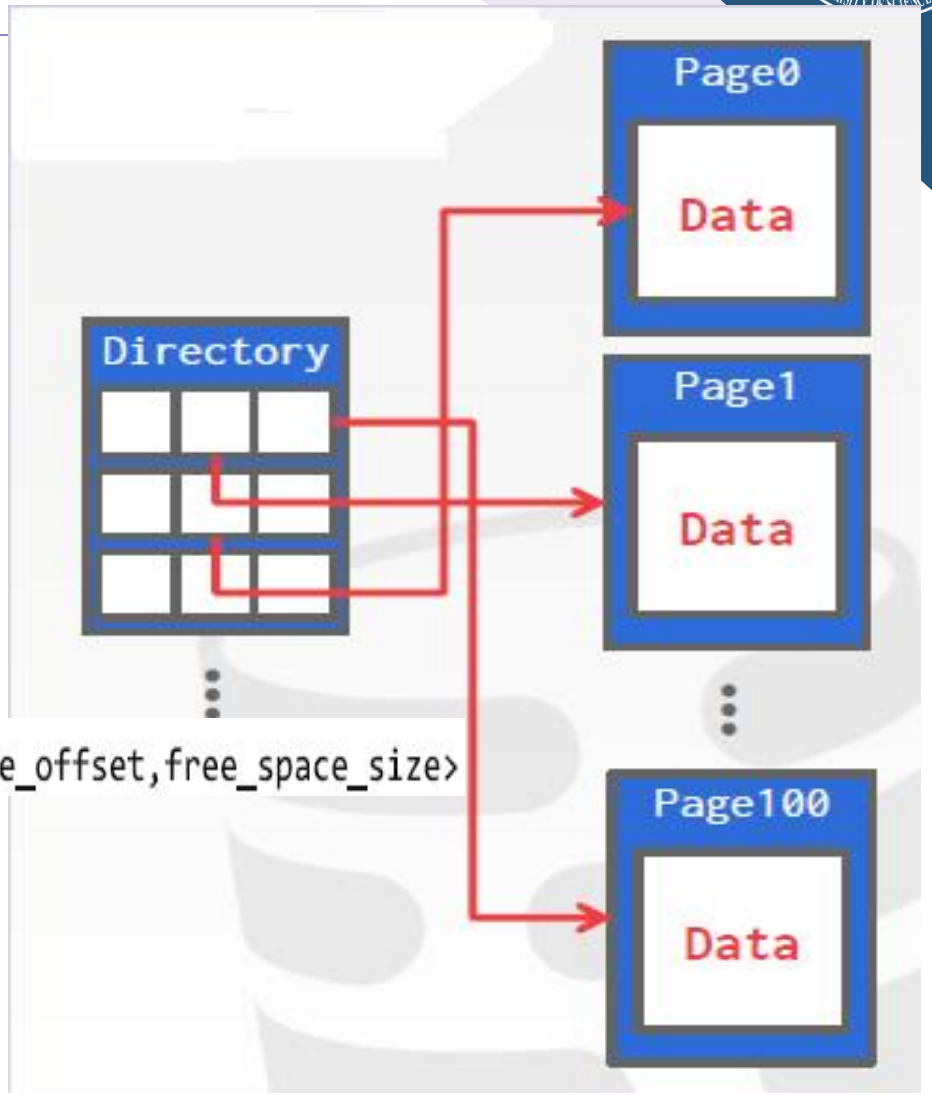
- 链表（Linked List）：以链表的形式将文件中的空闲页和数据页分别勾连起来。
- 堆文件头部设立一个header page，并存放两个指针，分别指向：
  - 空页列表（free page list）头部
  - 数据页列表（data page list）头部
- 每个page均记录当前空闲的空槽（slot）



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同步。

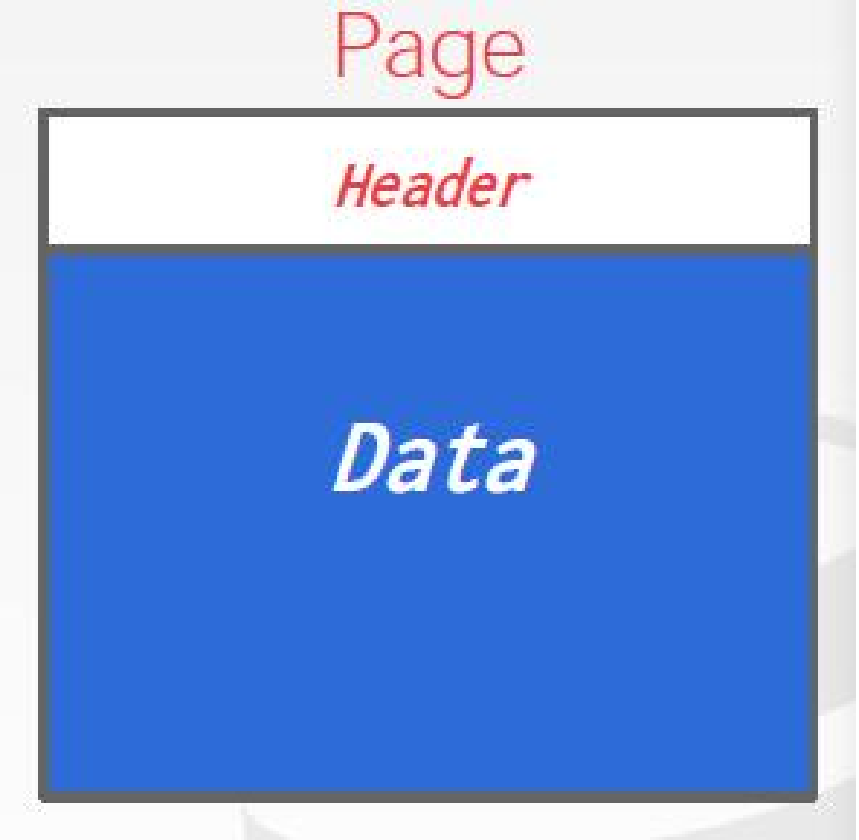


页的组织结构（Page Layout）

一个页面的内部结构，包括：

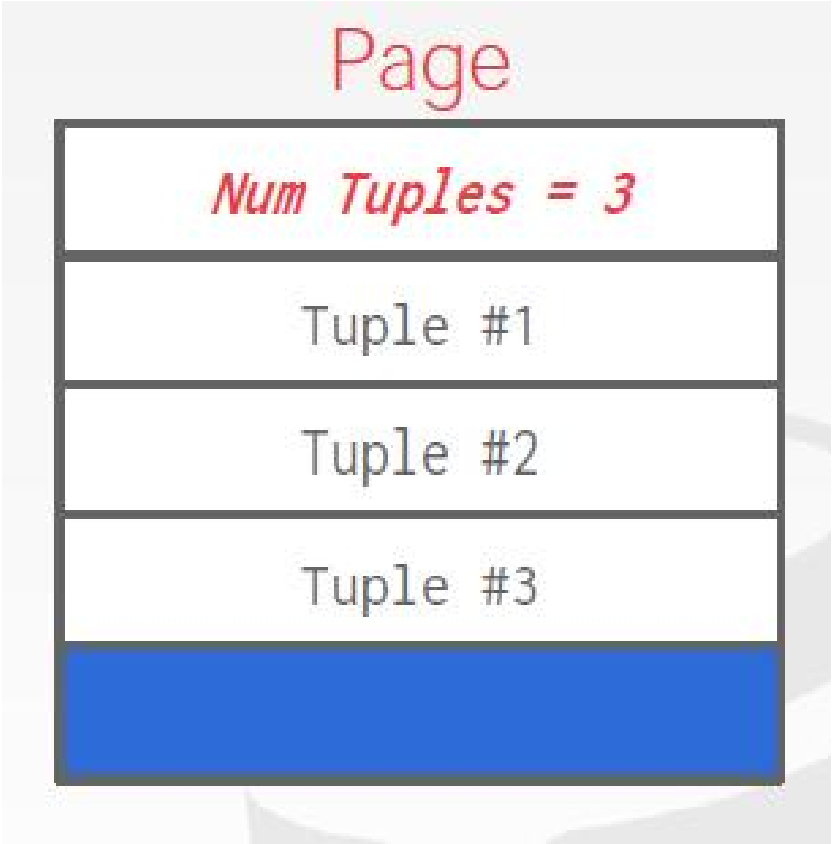
- 页头（page header），包含有关页内容的元数据信息：
  - 页大小
  - 校验和
  - DBMS版本
  - 事务可见性
  - 压缩信息

  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）

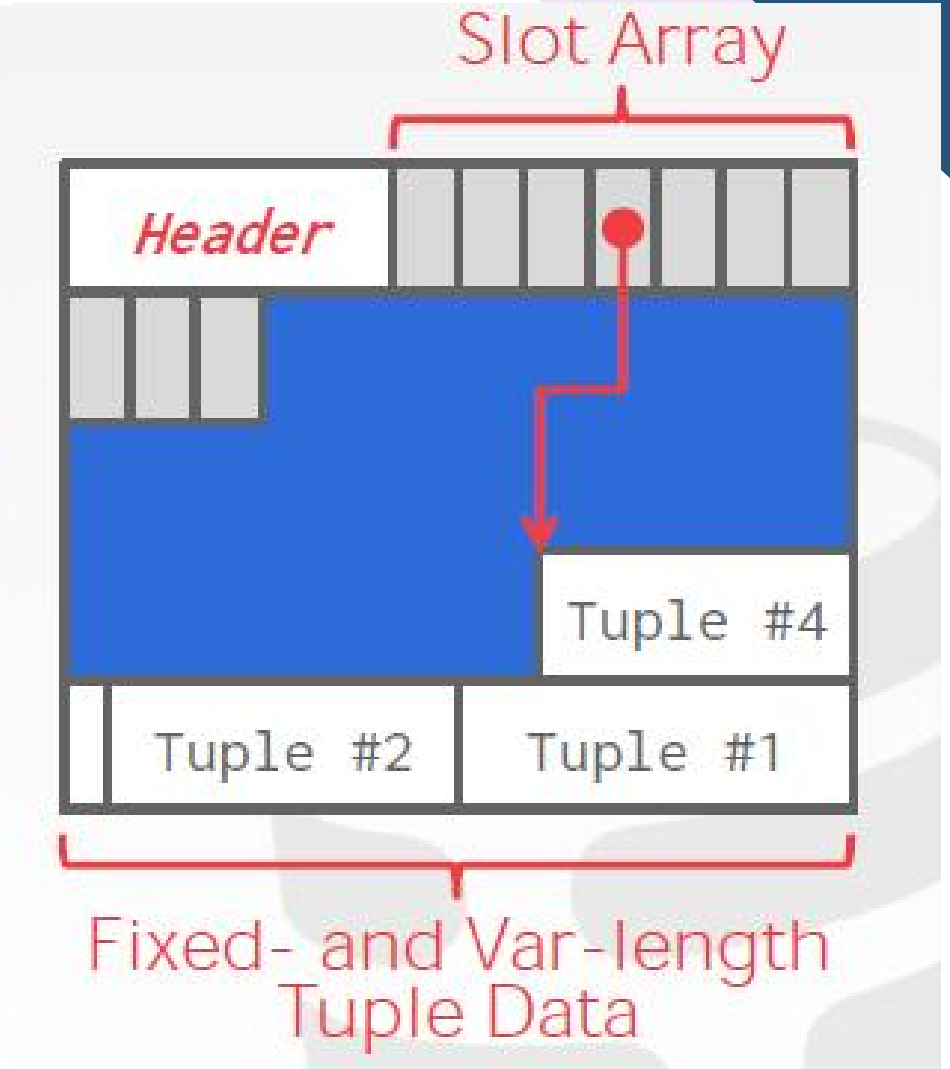
- Strawman Idea：
  - 记录页内的元组数，类似数组的方式进行存储；
  - 每次添加的元组放在已有元组的后面。
- 存在的问题：
  - 删除元组时会产生碎片
  - 变长元组可能产生其他更多问题，比如元组的查询开销。
- 一般用的较少，更常见的是slotted pages（槽页）方式



槽页（Slotted Pages）

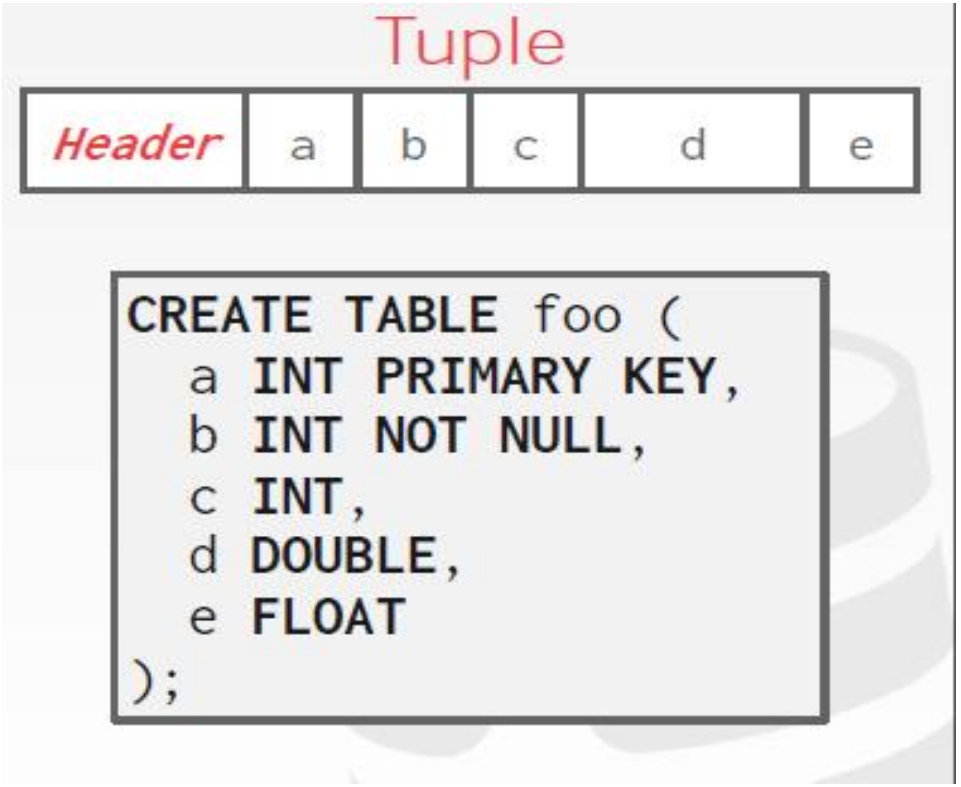
- 槽页：Slot数组将“槽位”映射到元组开始位置的偏移量。
- 当今DBMS最常用方法。
- Header记录：
  - 已占用的槽位；
  - 最后一次使用的槽的起始位置偏移；
- 元组：在页内倒序存放。
- 元组在内部的唯一标识符：

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



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。



元组设计

- 定长记录：全部由定长字段组成；
  - 定长记录的插入和删除易实现；
  - 注意：内存对齐问题
- 变长记录：允许记录中存在一个或多个变长字段。变长字段在记录中的偏移位置不确定；
  - 2种实现方式：
    - 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

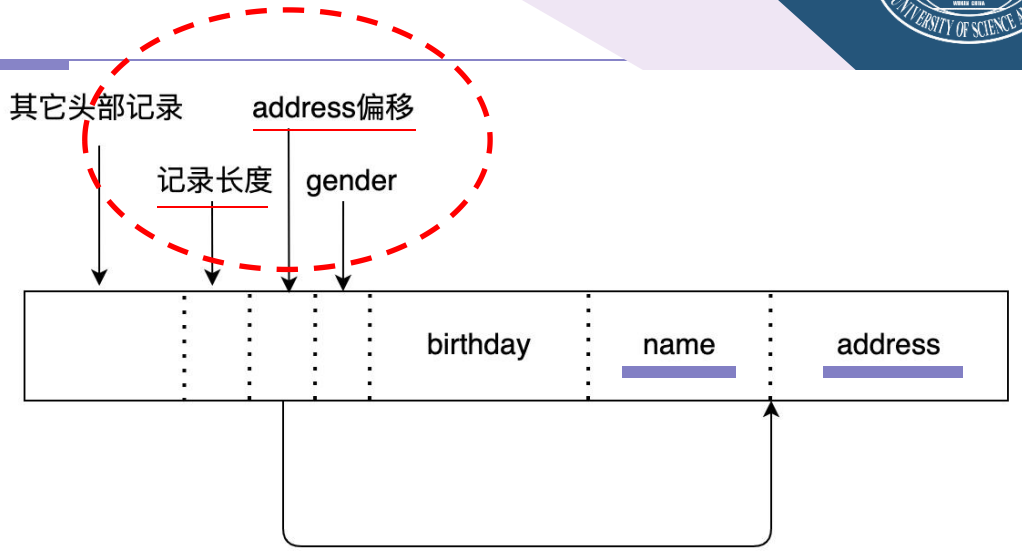


图1 变长记录表示方法一

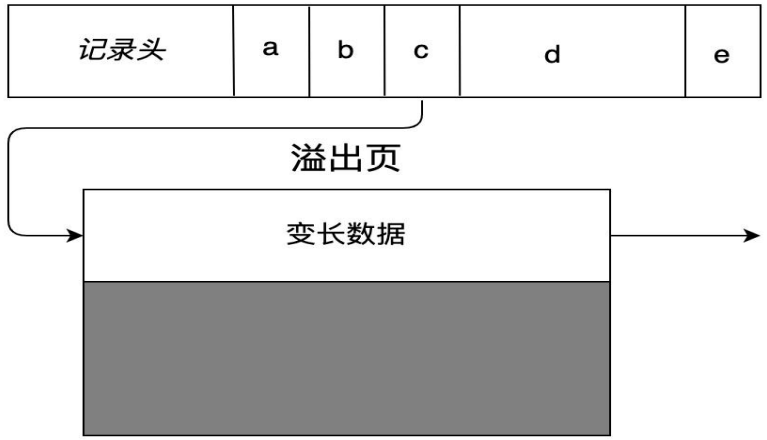
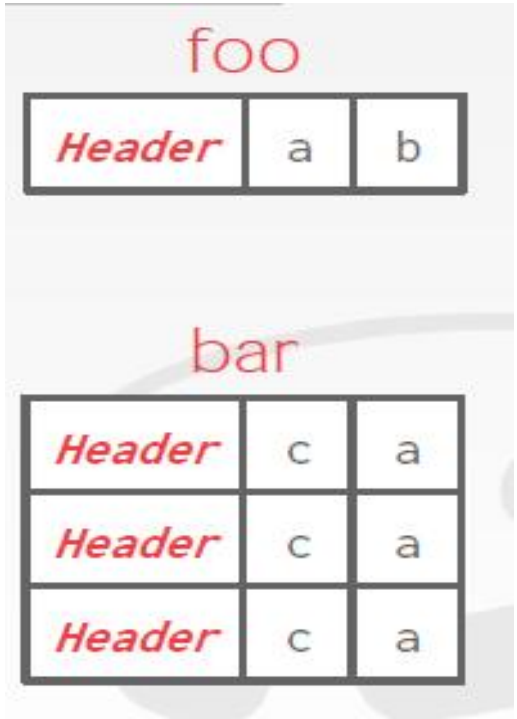


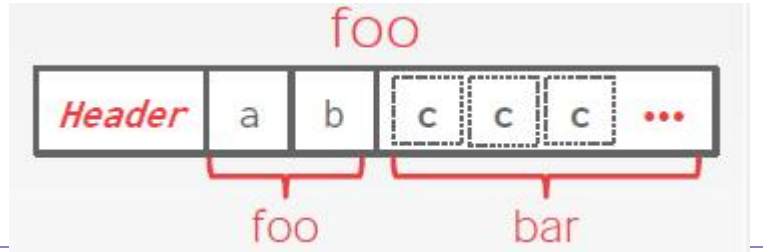
图2 变长记录表示方法二

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。

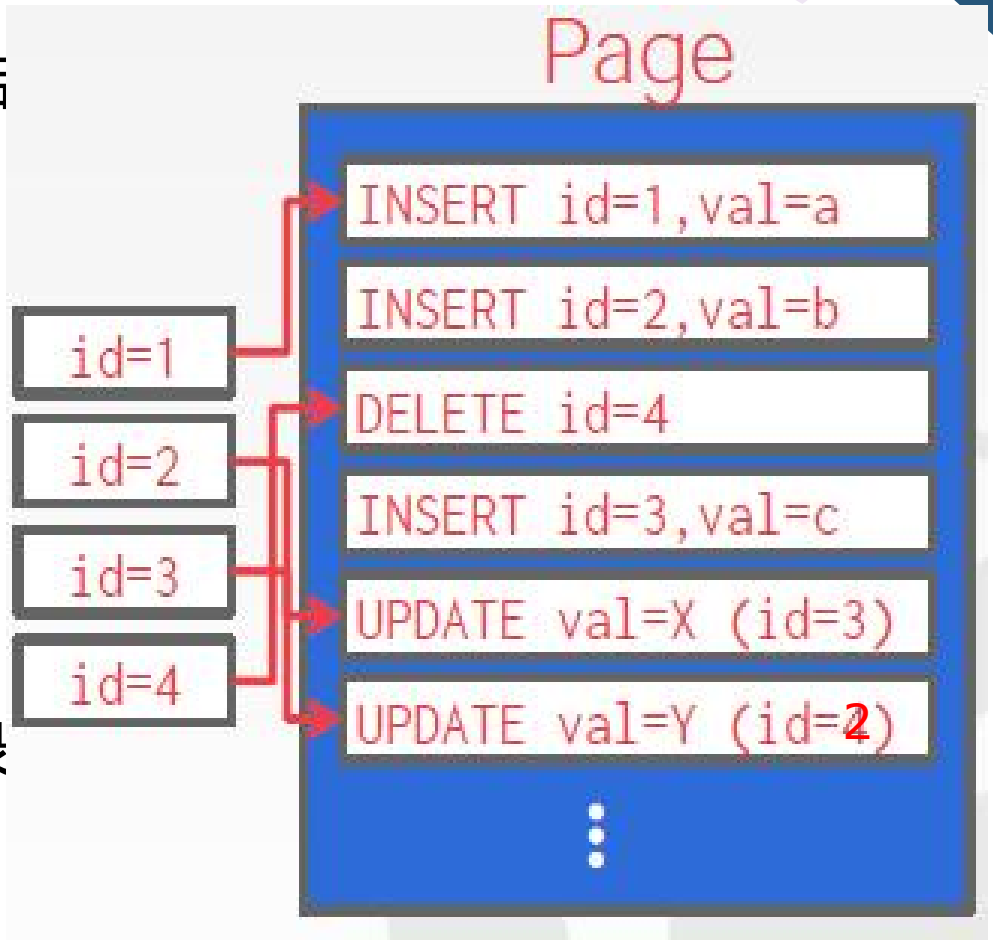
```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
CREATE TABLE bar (
  c INT PRIMARY KEY,
  a INT
  REFERENCES foo (a),
);
```





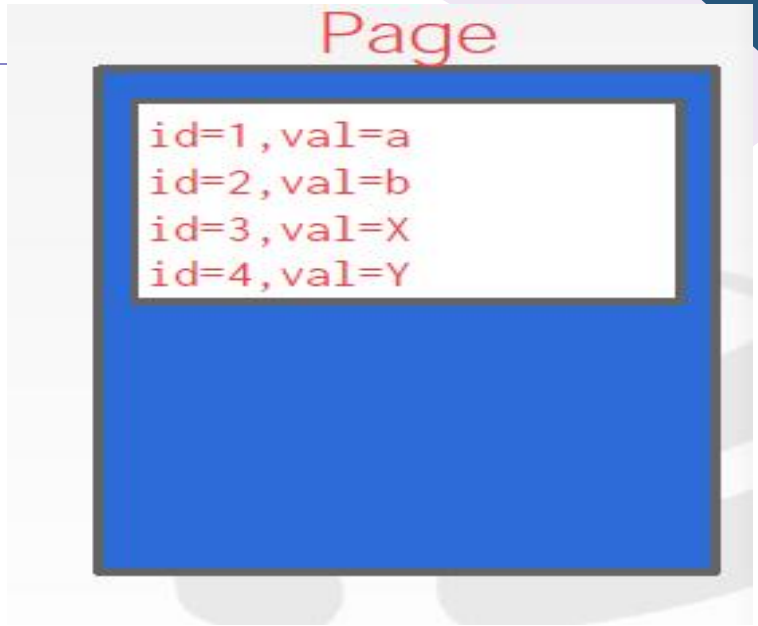
日志式(Log-Structured)文件组织

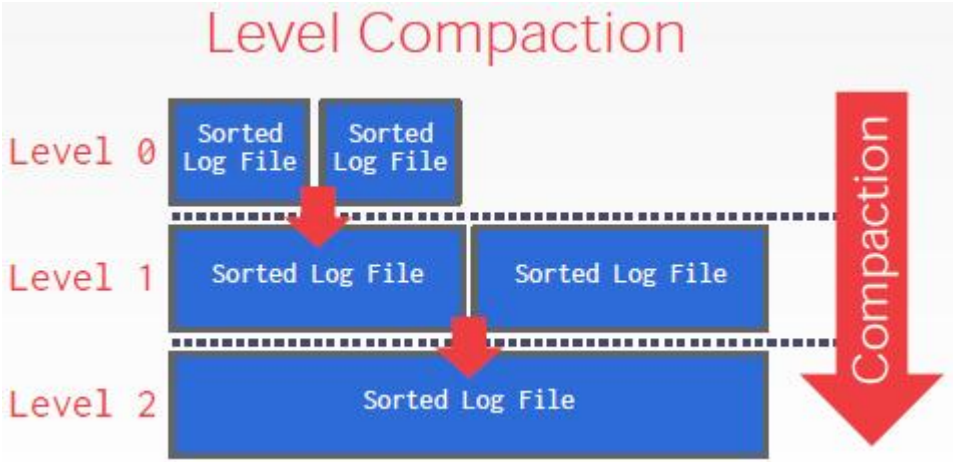
- DBMS不存储元组，而**只存储日志记录**
- 系统添加日志记录来反映数据库更新的结果
  - “插入”：存放整个元组；
  - “删除”：标记该元组被删除；
  - “更新”：记录被修改的属性的变化。
- 当需要读取日志记录，DBMS可以反向扫描日志，重新创建元组，还可“回滚”。
- 日志可定期压缩（通过删除不必要的记录来合并日志文件）。



日志文件组织

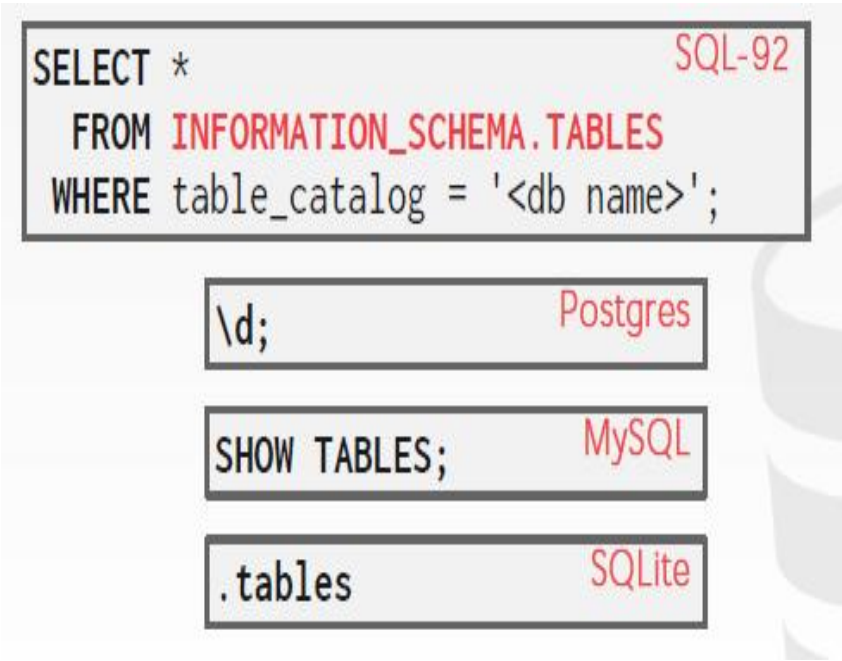
- 不同数据通常可能在不同page，如果将更新信息写到数据页，则需要访问多个page。
- 将数据更新信息写到一个或者连续页效率更高。
- 可建立“日志索引”，方便查找相关的日志记录。
- 定期压缩日志，去除不必要的记录。





8.1.3 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的数据库中，如SQL Server的master数据库。
- DBMS常会提供一些“非标准”的方法来检索这些系统目录表。

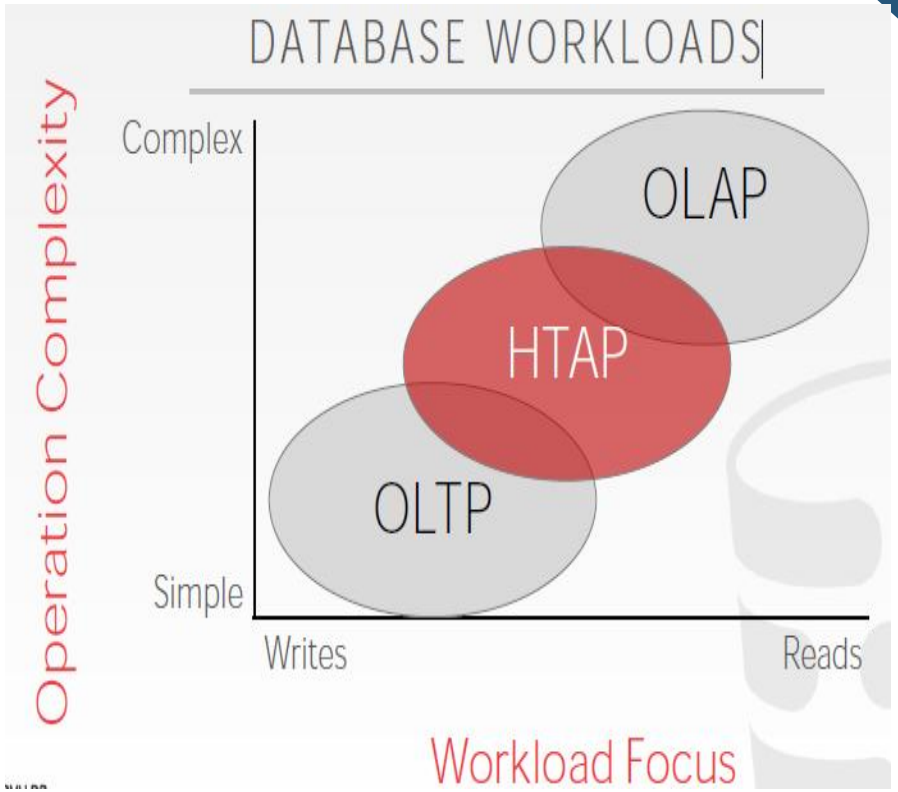
8.1.4 存储模型（Storage Model）

- 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。按工作负载类型区分：
  - 联机事务处理（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（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（HTAP）

    兼具OLTP和OLAP特征

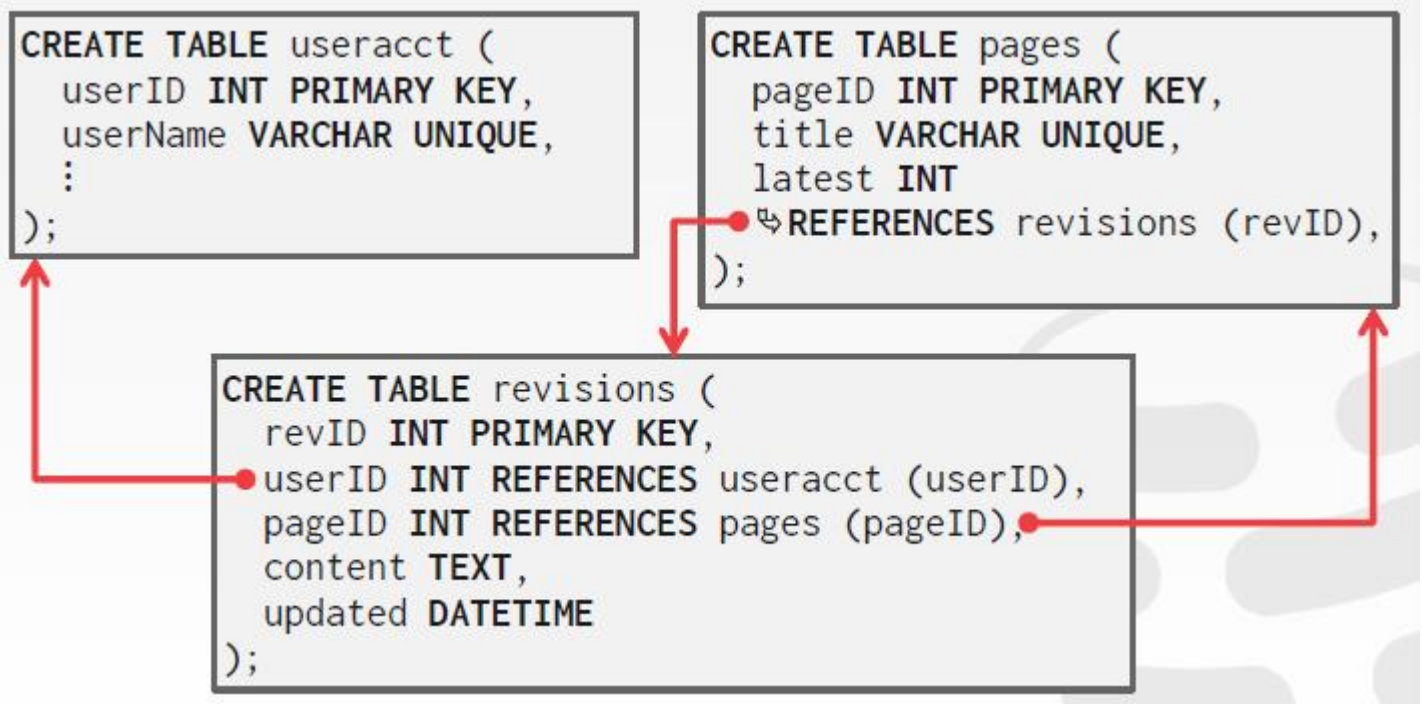


8.1.4 存储模型

- 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：
  - OLTP = 行存储
  - OLAP = 列存储

8.1.4 存储模型

维基百科例子



8.1.4 存储模型

- OLTP应用中，利用“读/写”SQL语句，实现业务计算。

- OLAP应用中，查询语句往往非常复杂，甚至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)
```

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revisions AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

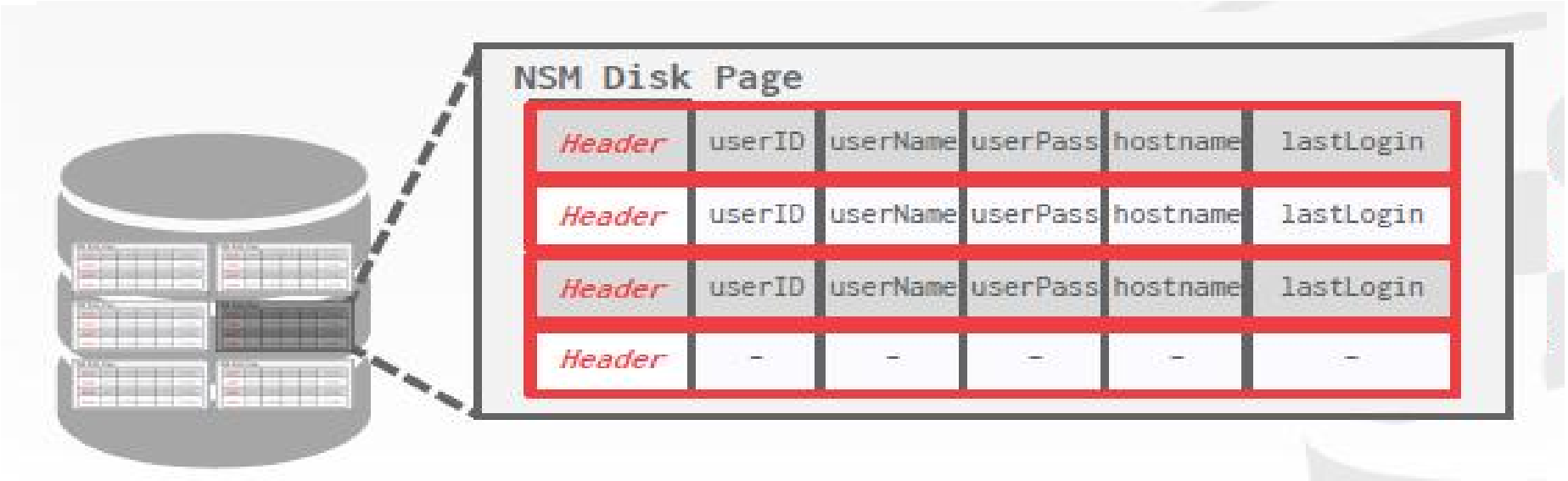
```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
INSERT INTO revisions
VALUES (?,?…,?)
```

8.1.4 存储模型

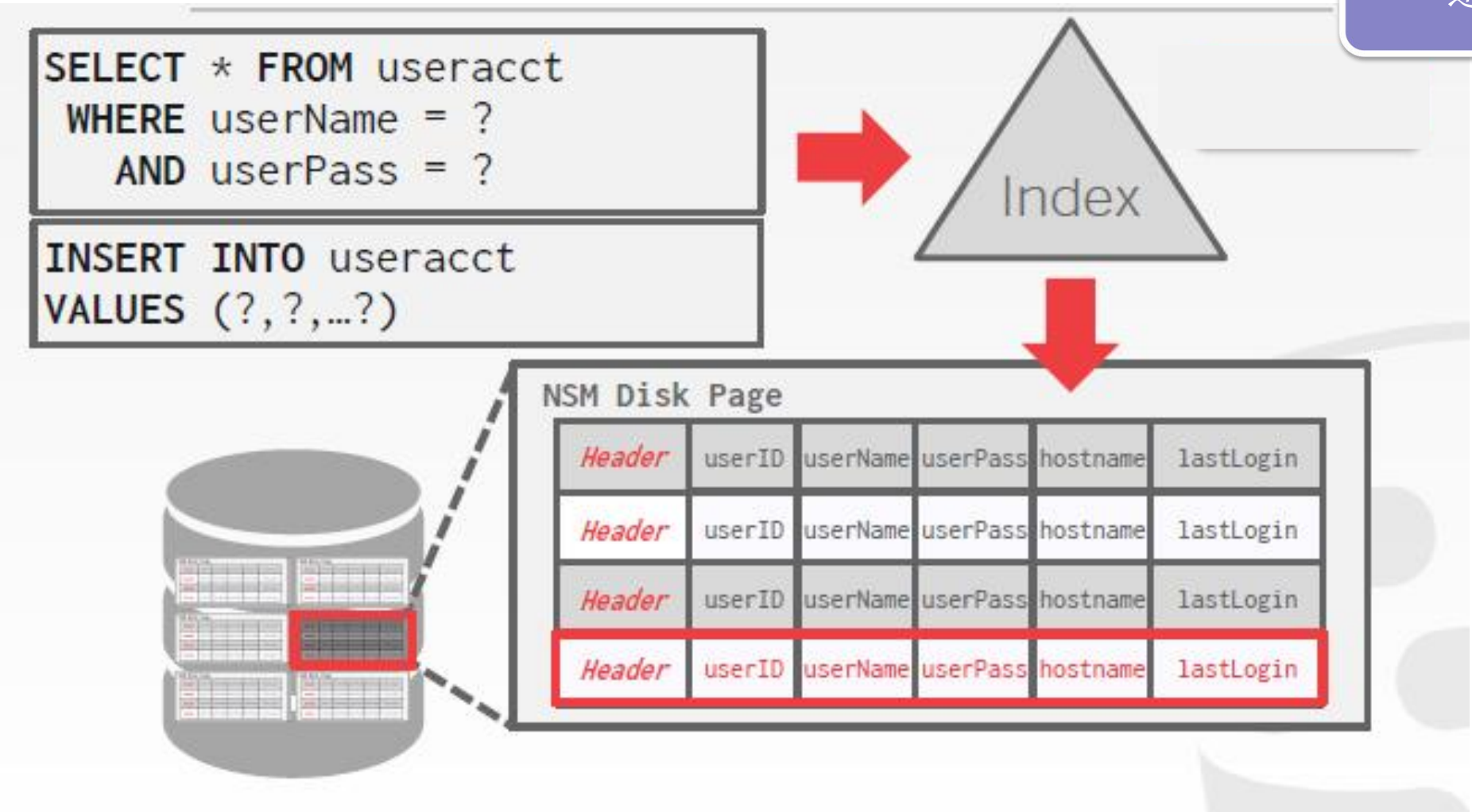
为适应OLTP或OLAP不同的工作负载，DBMS可采用不同方式进行元组存储。

- NSM（n-ary storage mode，“行存储”）
  - 行存储模型非常适合OLTP。
  - 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。



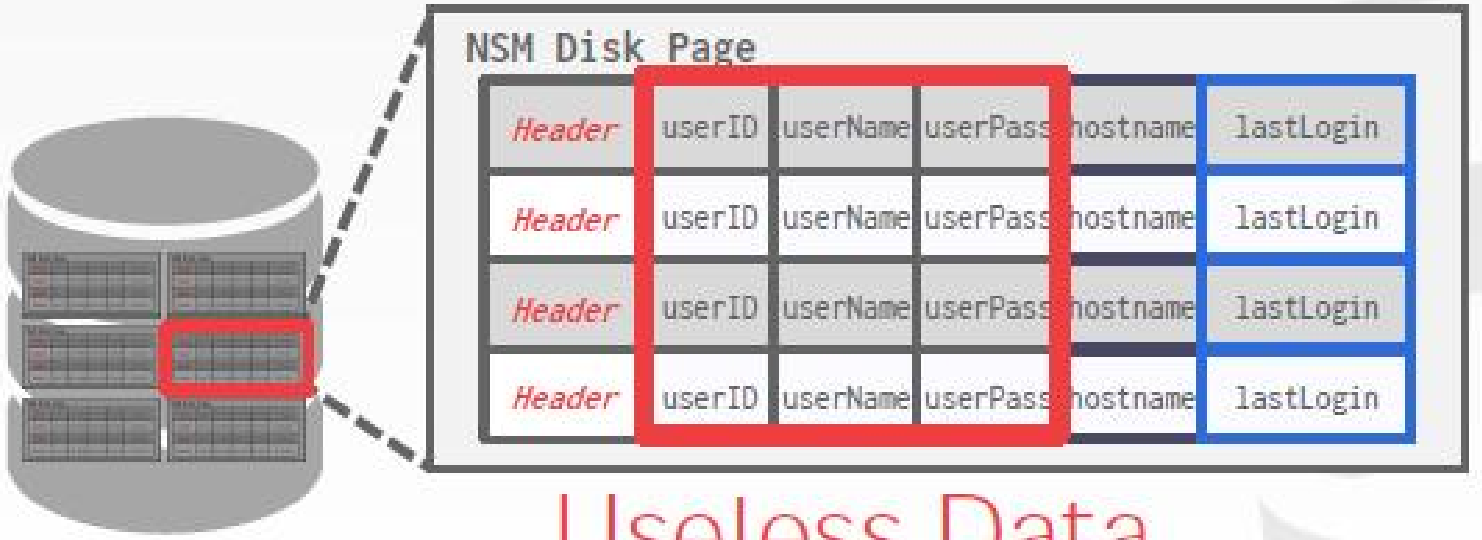
NSM（行存储）

适用


SELECT * FROM useracct
WHERE userName = ?
AND userPass = ?
INSERT INTO useracct
VALUES (?,?,…?)
Index
NSM Disk Page
Header userID userName userPass hostname lastLogin
Header userID userName userPass hostname lastLogin
Header userID userName userPass hostname lastLogin
Header userID userName userPass hostname lastLogin

NSM

```
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)
```

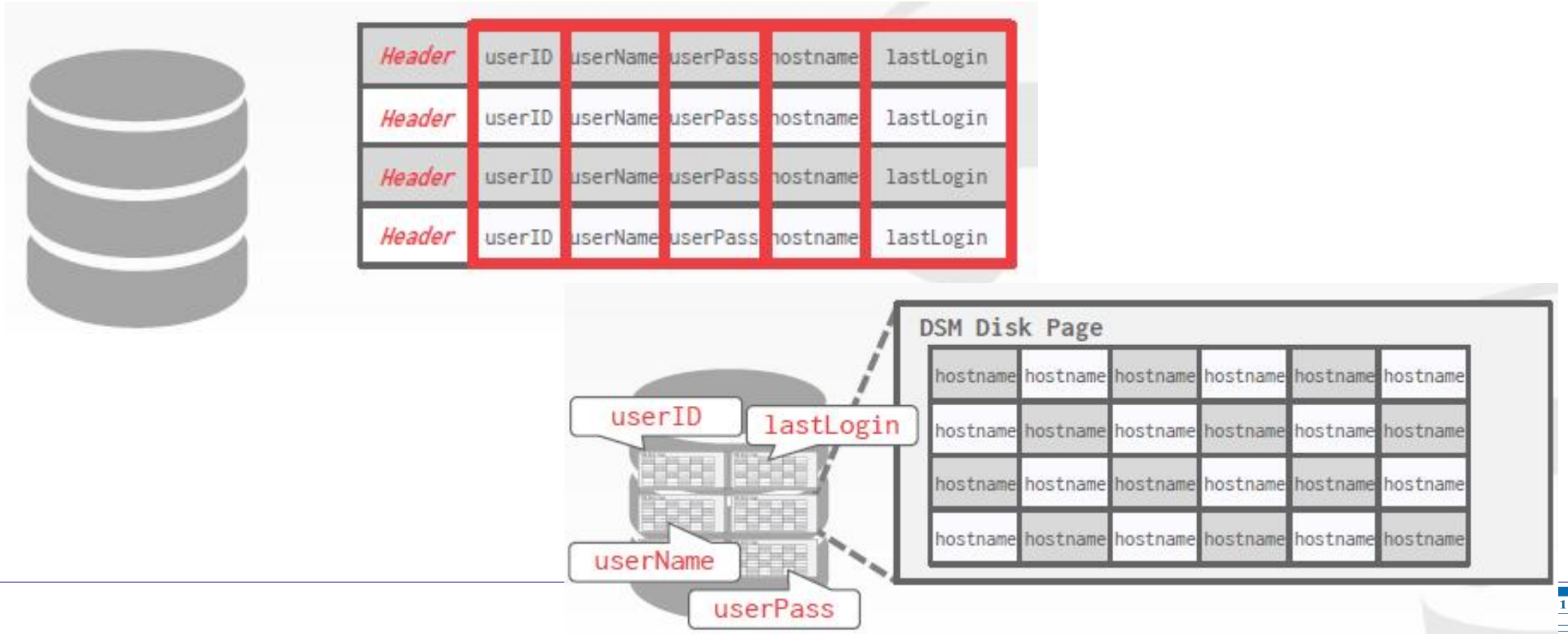


NSM：

- **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的大量部分属性，且伴随复杂查询语义时不适合。**

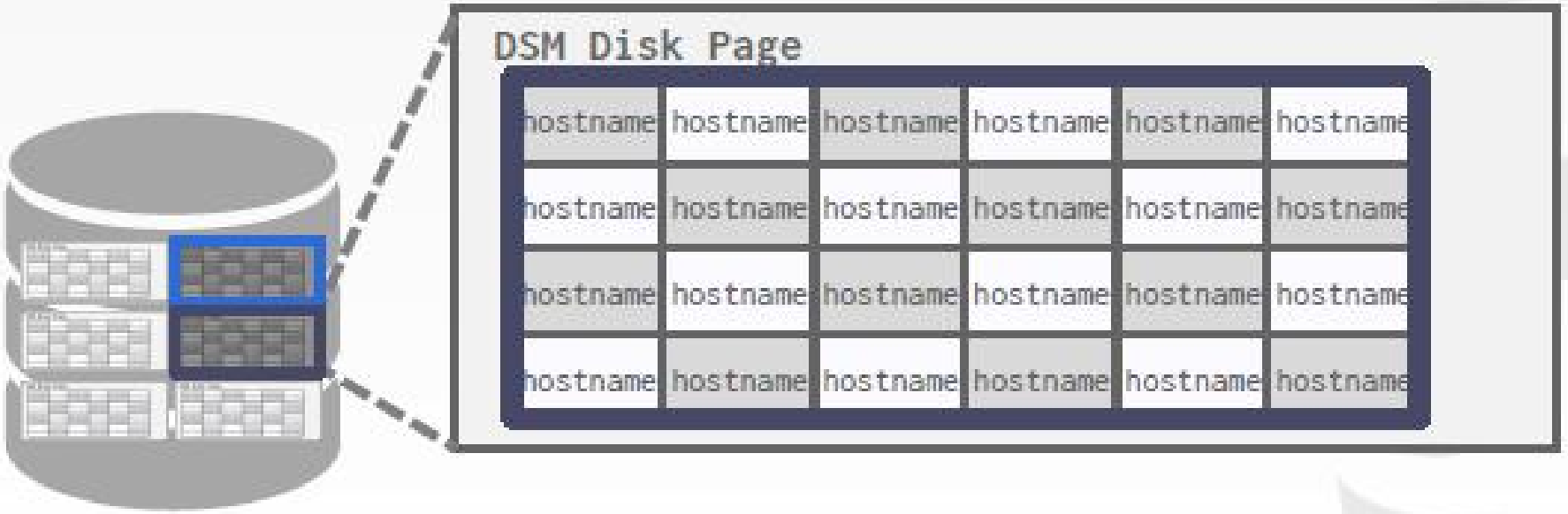
DSM（列存储）

- DSM（Decomposition Storage Model, 列存储）
  - DBMS将单个属性的值连续的组织在一个page中，按列存储；
  - 更适合OLAP，可以很好的适应大数据量、复杂查询语义、高负载查询。



DSM

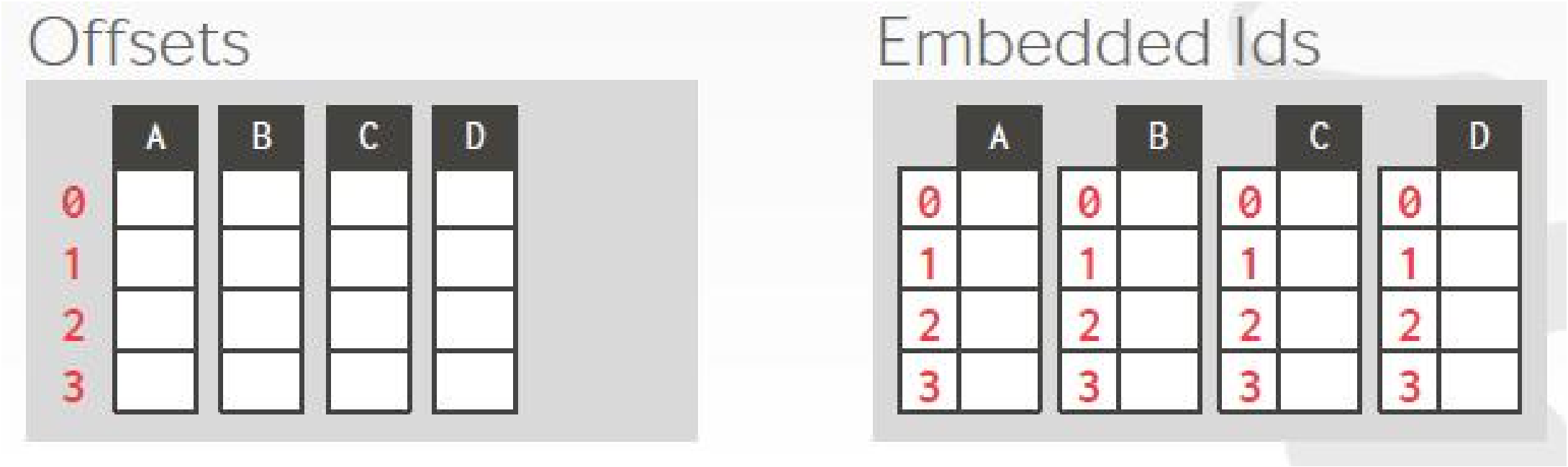
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



DSM

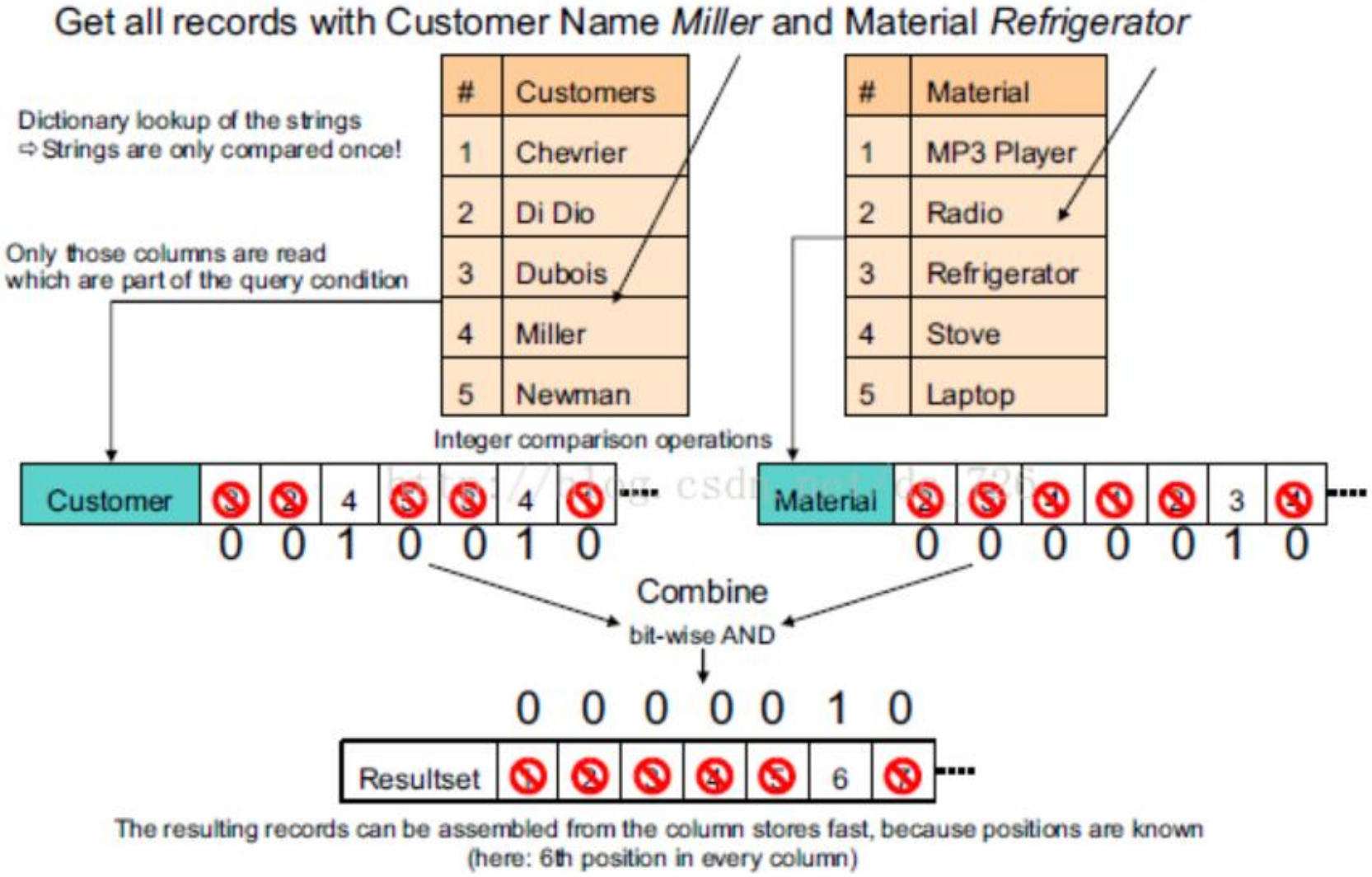
DSM中，如何进行“元组标识”？

- 选择1：固定长度偏移，对某个属性每个值具备相同宽度
- 选择2：元组ID嵌入，每个值与其元组ID一起存放



DSM

- 通过一条查询的执行过程说明列式存储(以及数据压缩)的优点：



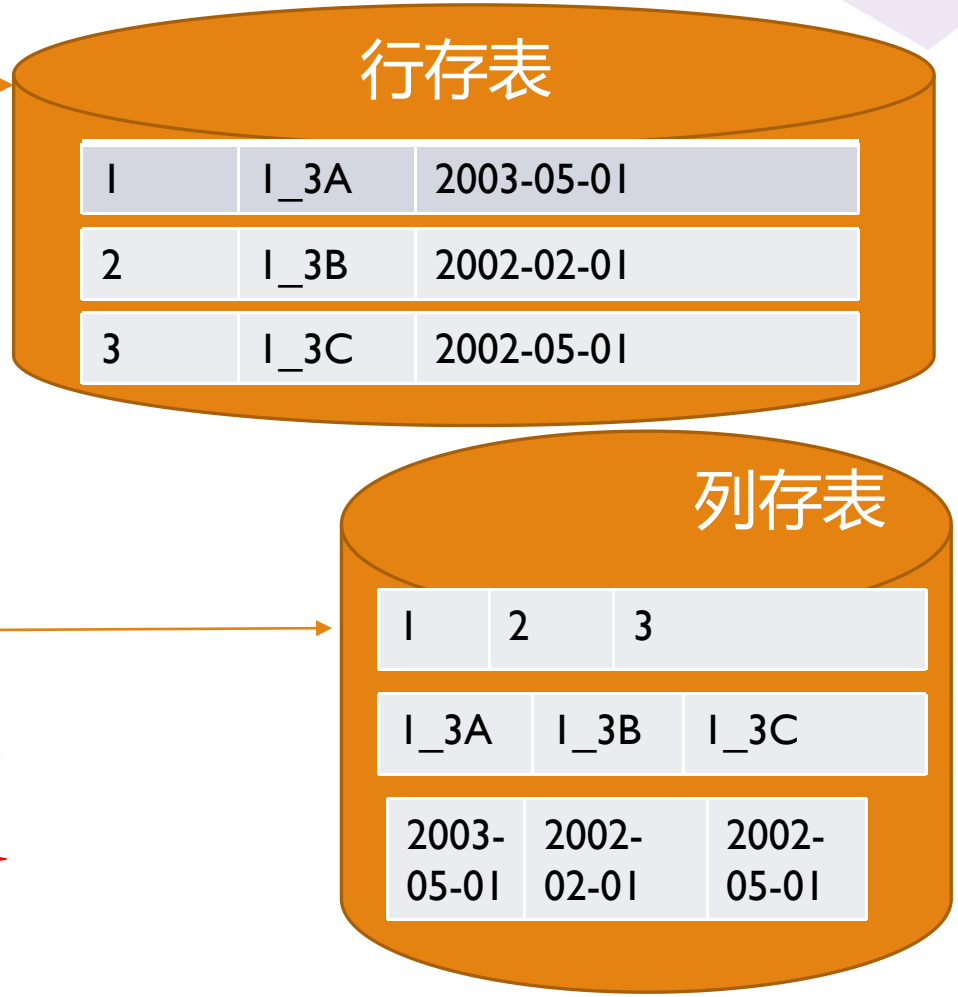
DSM

- DSM优点:
  - 由于只读取需要的数据，因此减少了对无用数据的I/O；
  - 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

数据库行列存储机制在解决社会性突发问题起到的作用

openGauss 行存&列存

Cust_no	Seat_id	Birth_date
1	1_3A	2003-05-01
2	1_3B	2002-02-01
3	1_3C	2002-05-01



疫情期间，如何排查火车上的新冠检测呈阳性乘客的周围乘客?