关系数据库引擎基础

本章主要内容

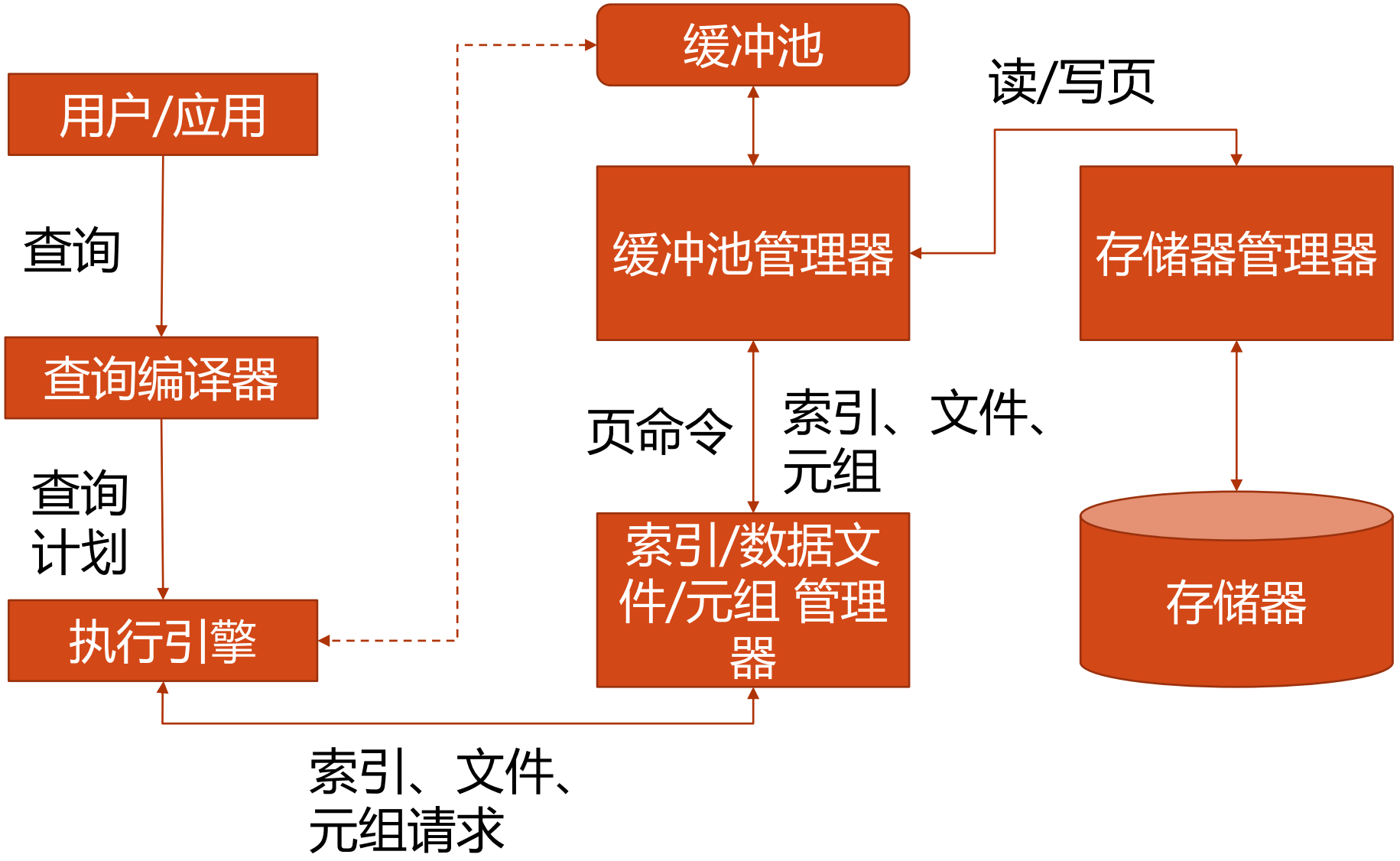
1 数据库存储结构

2 B+树索引

3 缓存

4 查询处理

查询和关系数据库引擎


缓冲池
用户/应用
读/写页
查询
缓冲池管理器
存储器管理器
查询编译器
页命令
索引、文件、
元组
查询
计划
索引/数据文件/元组 管理器
存储器
执行引擎
索引、文件、
元组请求

数据库存储结构

本节主要内容

1 数据库存储结构概述

2 堆文件

3 页设计（槽页）

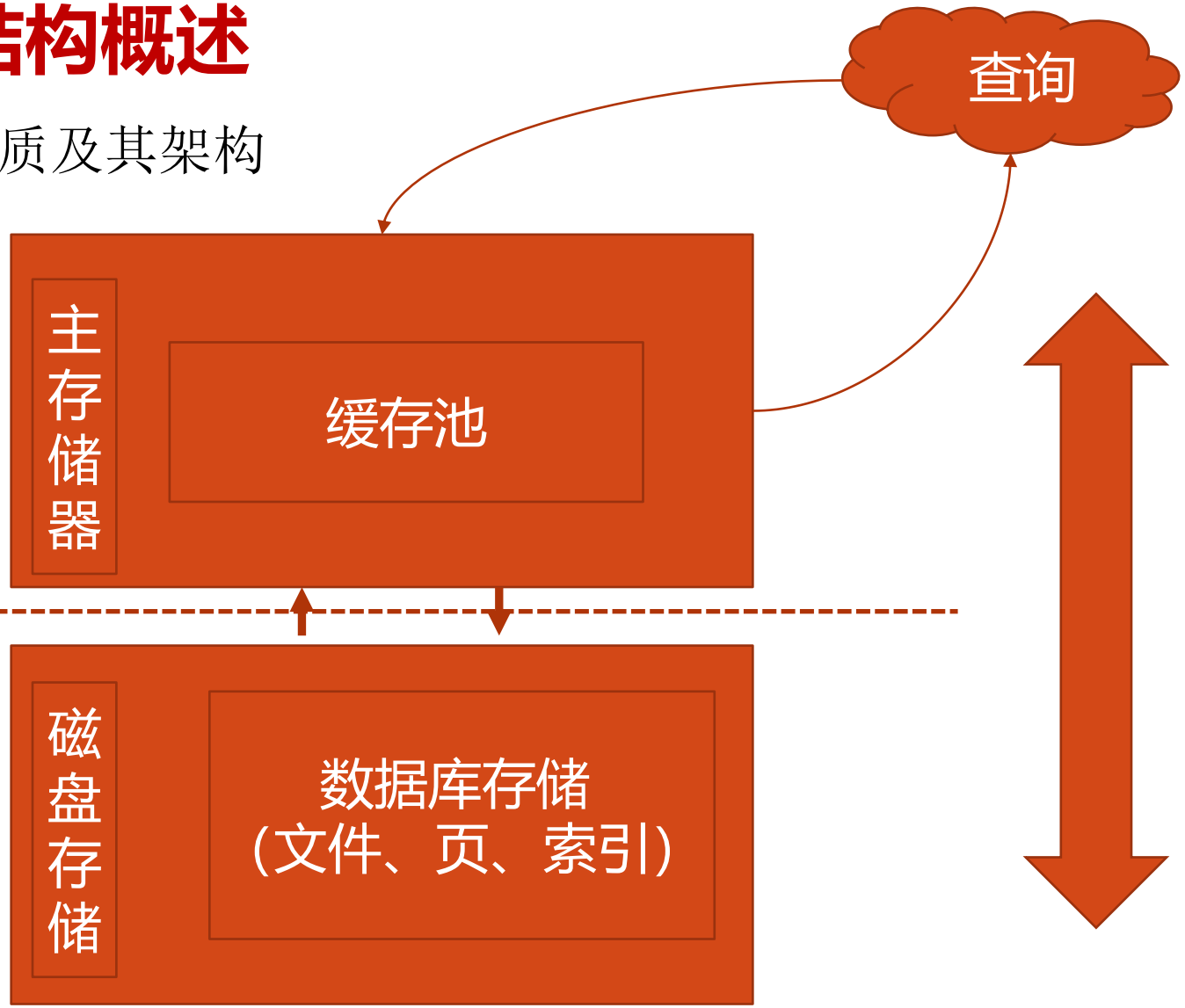
4 元组设计

5 存储模型简介

1 数据库存储结构概述

用于数据库的存储介质及其架构

- 易失
- 高效随机访问
- 字节可寻址



- 非易失
- 顺序访问
- “块”可寻址

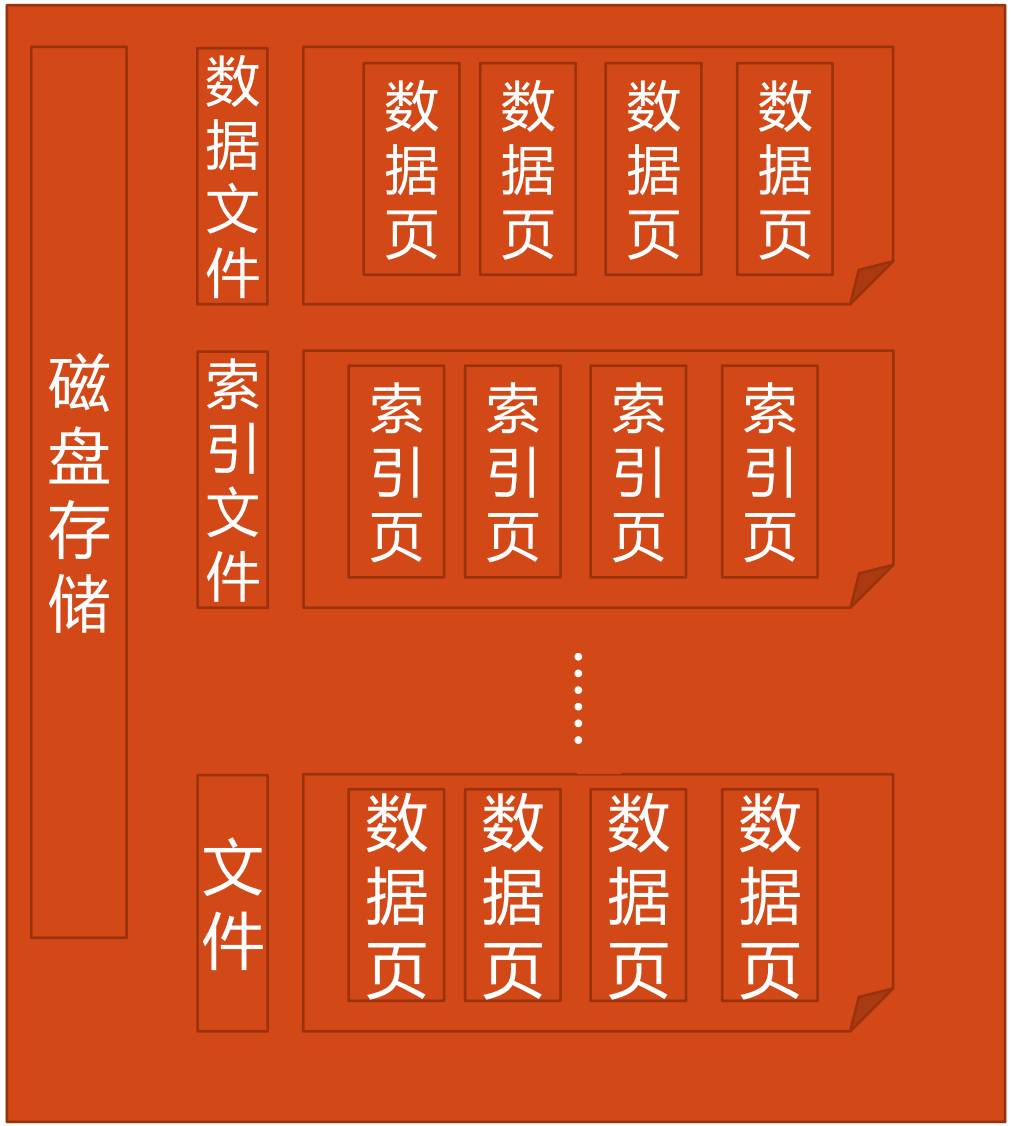
访问速度快
容量小
昂贵

访问速度慢
容量大
便宜

1 数据库存储结构概述

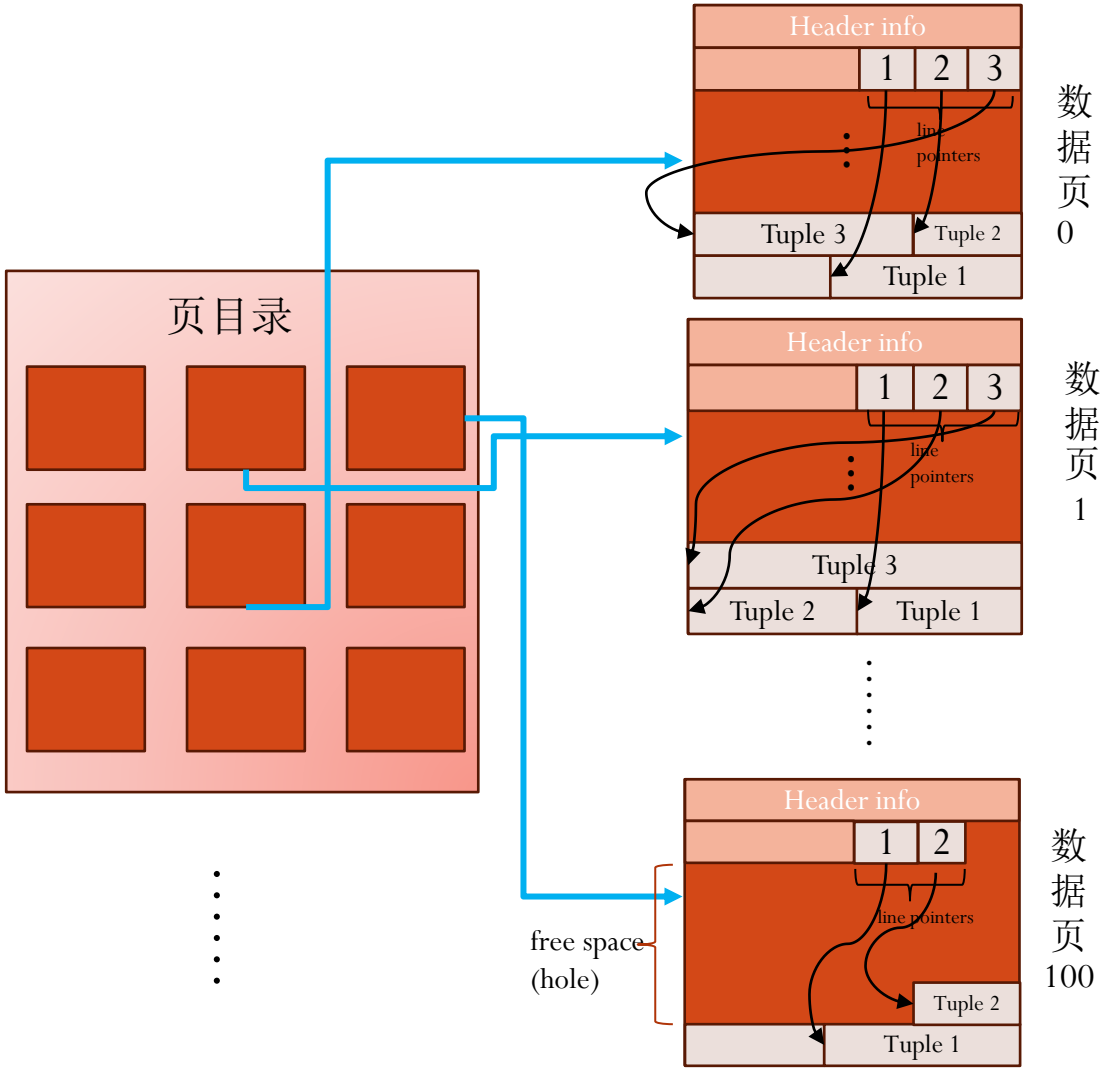
为了有效的进行数据库文件管理，**一般**将文件组织为“页”的集合。

- 追踪页面数据的读写操作
- 追踪可用的存储空间



2 堆文件

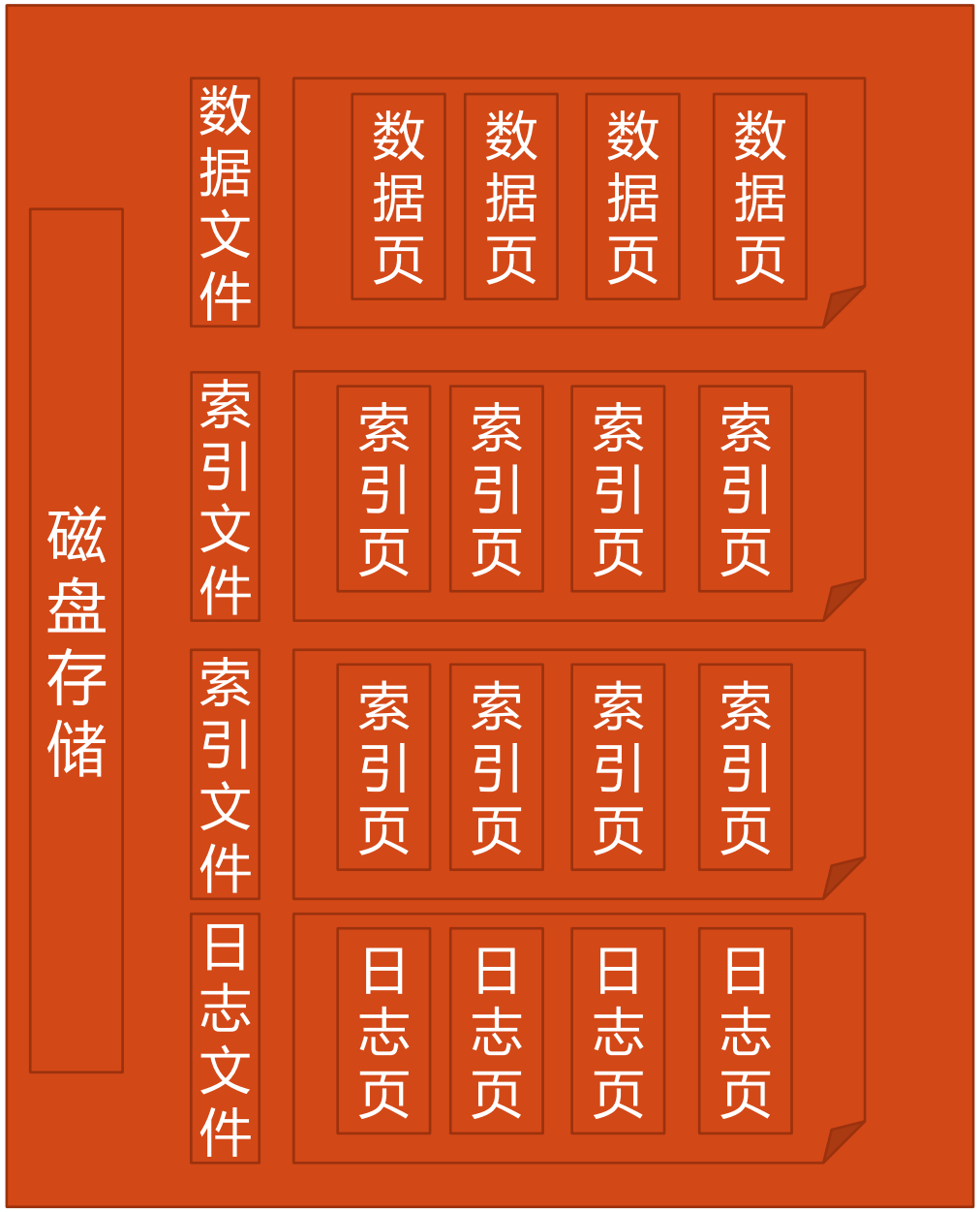
- 不同的DBMS管理磁盘中pages的方式不同，堆文件组织（Heap File Organization）是一种常见的方式。
- 堆文件中设立一类专门的页面（目录页），用于记录所有的数据页的存放位置，同时记录每个页面的空闲空间信息。
- DBMS必须保持目录页与所有页的当前信息同步。



3 页设计（Page Layout）

- 数据库的页可以容纳：元组、元数据、索引、日志记录等。数据一般不混合存放。
- 数据库的页具备固定大小（如4K），一个数据库页是硬件页面的整数倍。
- 每个页具备一个唯一页面ID（寻址）：PageID

一般情况下为了简化设计，直接使用文件+页在文件内的偏移地址作为页的PageID。



页头（Page Header）

每个页面都分为两部分：页头和数据区域。

- 页头（page header），包含有关页内容的元数据信息：如（页大小，校验和），与事务相关的标记如脏数据等。
- 数据区域：面向元组型、面向日志型.....

基于数组的面向元组型的页设计

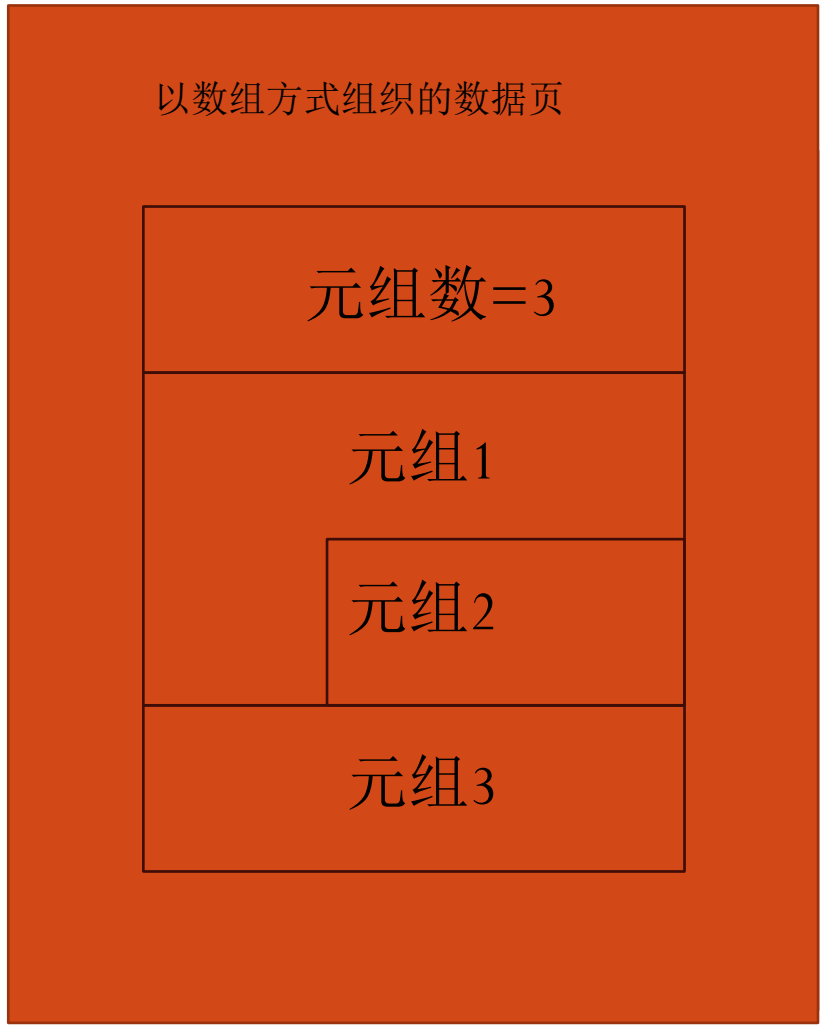
按照数组思想：

- 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

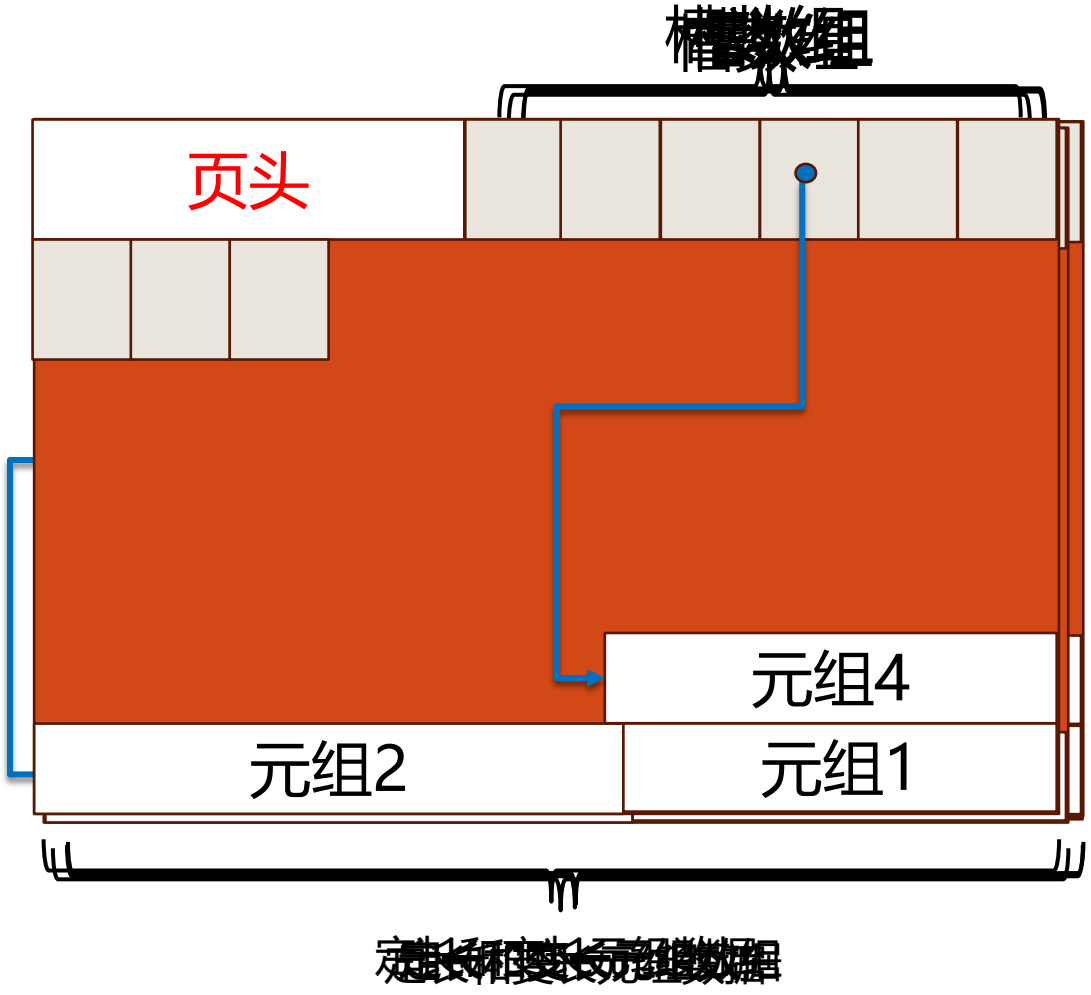
元组1的长度是变长的情况



基于槽页的面向元组型的页设计

按照槽数组的思想：

- 槽数组将“槽位”映射到特定元组开始位置的偏移量
- 元组在页内倒序存放。
- 页头记录已占用的槽位以及上一次使用槽位的开始位置。
- 元组在内部的唯一标识符（TupleID）：可以使用page id和slot id（或偏移量），也可包含文件位置信息。



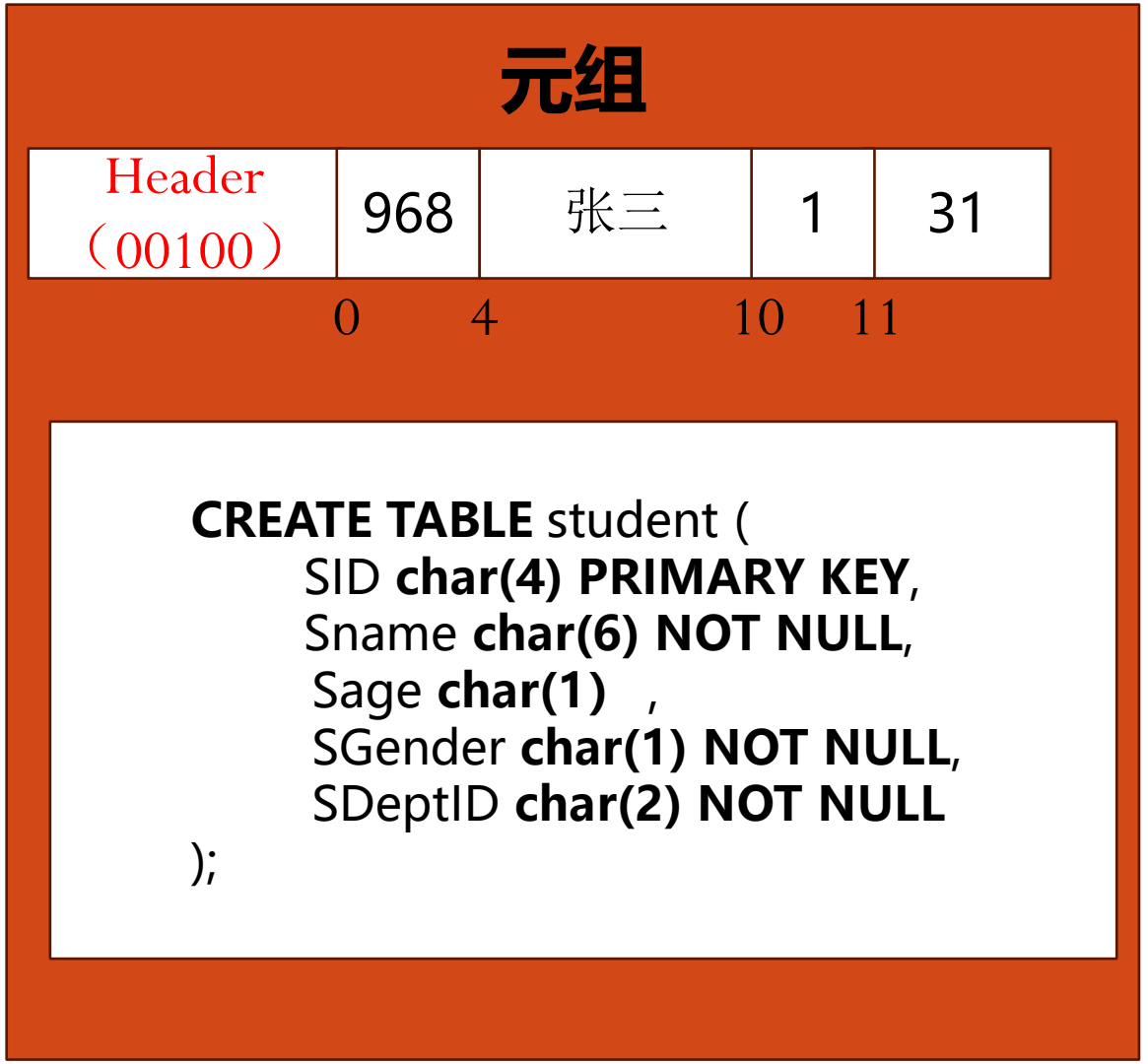
特点：

- 定长、变长元组轻松应对 。

4 元组设计

“字节序列”：一个元组在页中本质上是一个字节序列

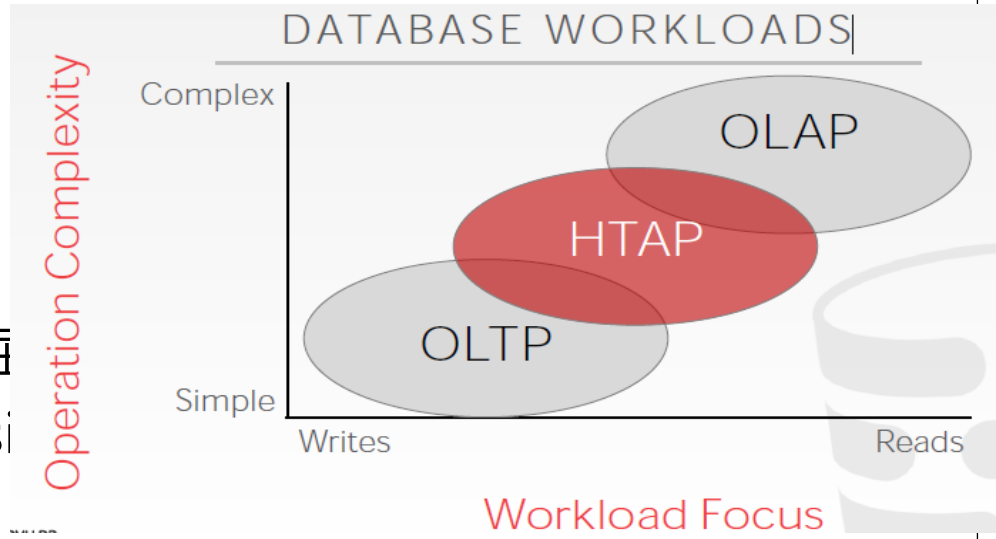
- DBMS负责将这些字节解释为各个属性的类型和值，页中无需存放关系模式信息，专门的字典页（catalog page）可有效减少重复信息。
- 每个元组有一个前缀为header包含元数据（例如对创建元组的事务号、空值的Bit Map）



5 存储模型概述（Storage Model）

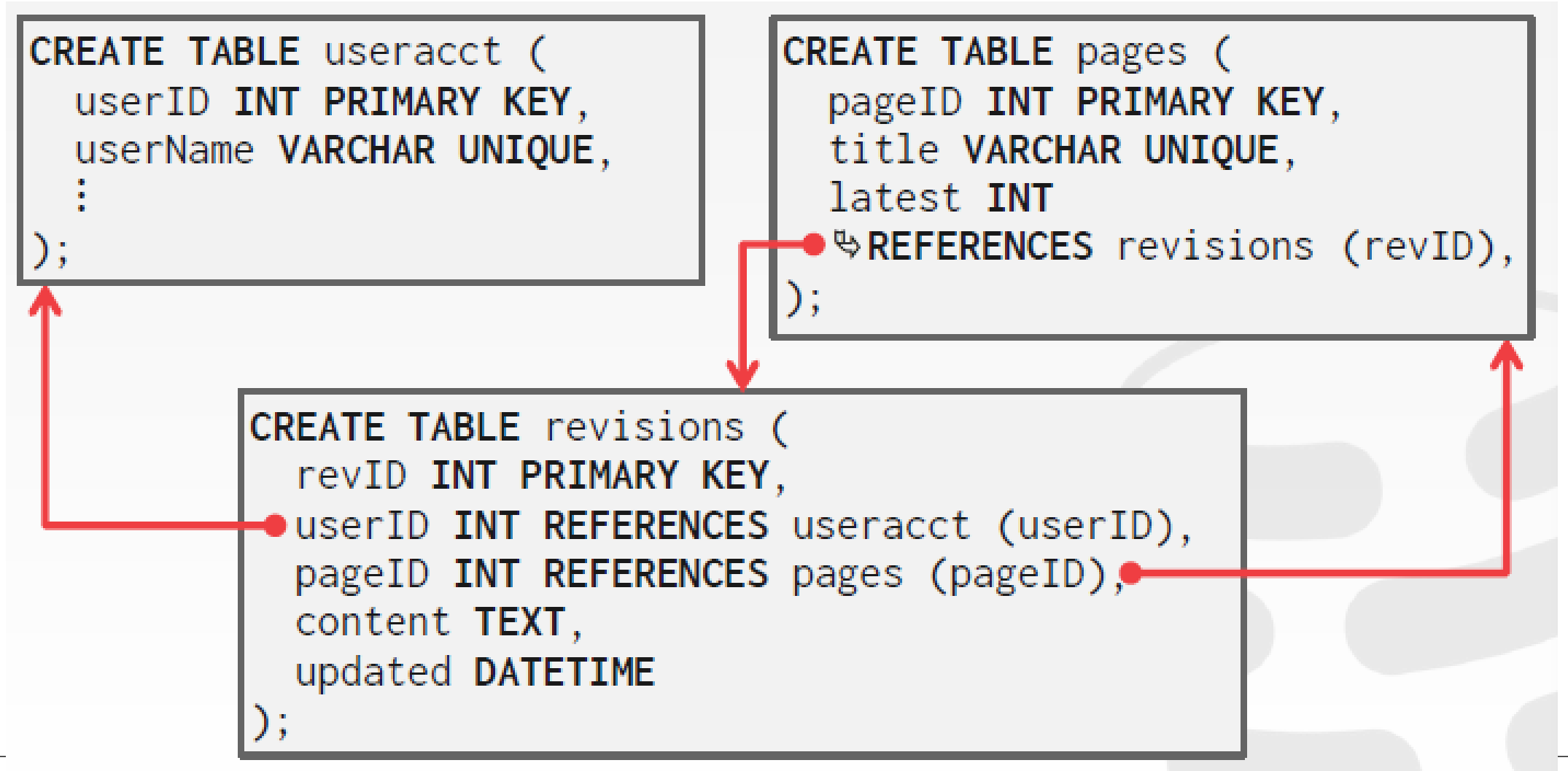
数据库的存储模型从全局、应用特征等角度，尤其大数据环境，考虑数据库如何适应需求。这里考虑的“工作量（Workloads）”如下：

- 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 Processi HTAP）
  - 兼具OLTP和OLAP特征



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
);

存储模型

当应用在OLTP中，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

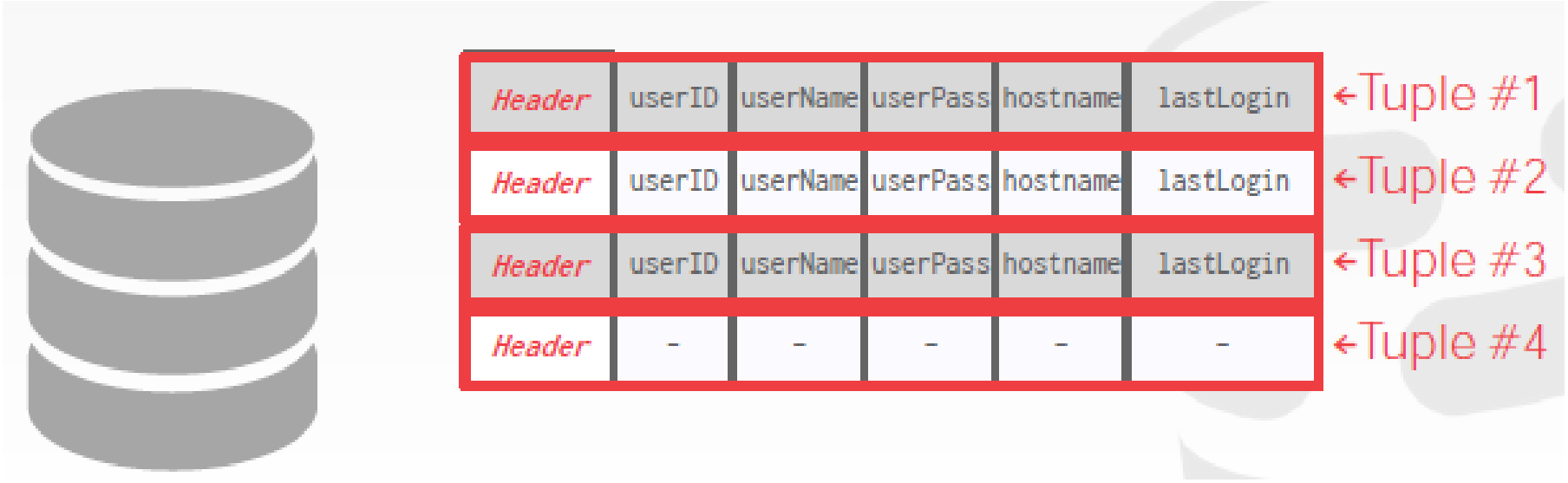
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM
               U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY
  EXTRACT(month FROM U.lastLogin)
```

NSM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。

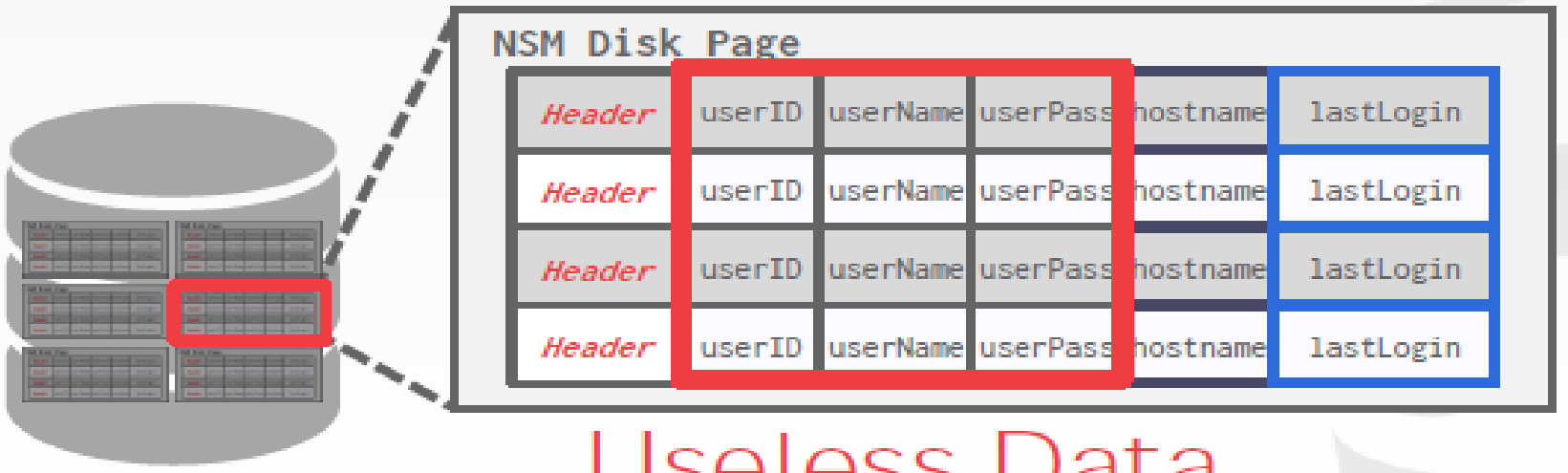
常见的n元存储模型（n-ary storage mode，NSM，又名“行存储”）非常适合OLTP。此时，单个元组的所有属性连续的分布在一个page中，查询往往涉及单个实体（工作量较少），并能适应较为繁重的“更新”工作量



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)
```



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。

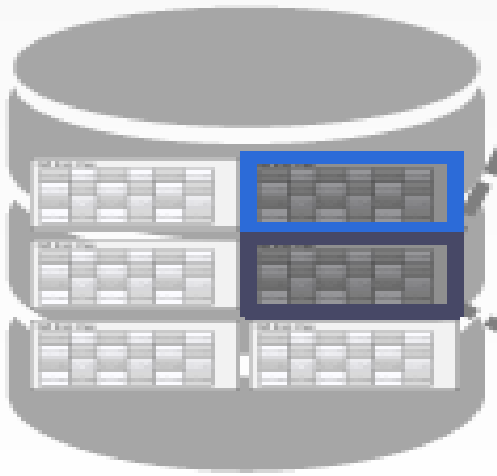
DSM

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