Introduction

DBMS的系统性能，从存储层和计算层两个方面考虑的话，分别体现在数据存储在磁盘上的效率，以及把数据移动到CPU寄存器上的效率。

列式存储的layout：

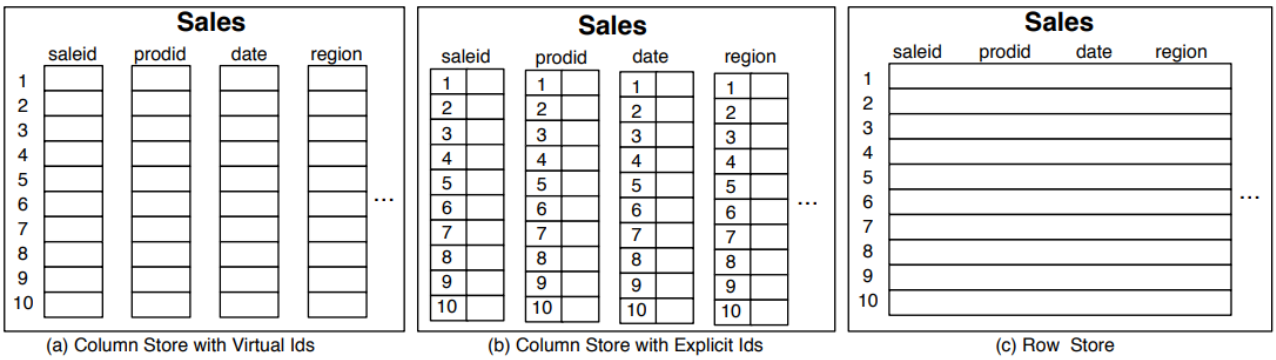


Figure 1.1: Physical layout of column-oriented vs row-oriented databases.

在OLAP的分析型数据库中，通常只需要读取某几列数据而不是全部列的数据，如果使用行存，会存在比较严重的读放大问题。进而也在把memory的数据移动到CPU寄存器的时候，减少了所要移动的数据量，提高了IO效率与内存bandwitdh的使用率

Column-Store Architectures Features

- Virtual ID

如上图的b，每一列的每一行数据都显式存储着一个ID，而a则是使用该数据在该列的偏移量作为虚拟ID。

那么对于固定宽度的列来说，访问列A中的第i个数据，就只需要直接访问startOf(A) + i * width(A)位置的数据即可，没有其他什么间接引用乱七八糟的东西。

- 以块组织数据&向量化执行引擎

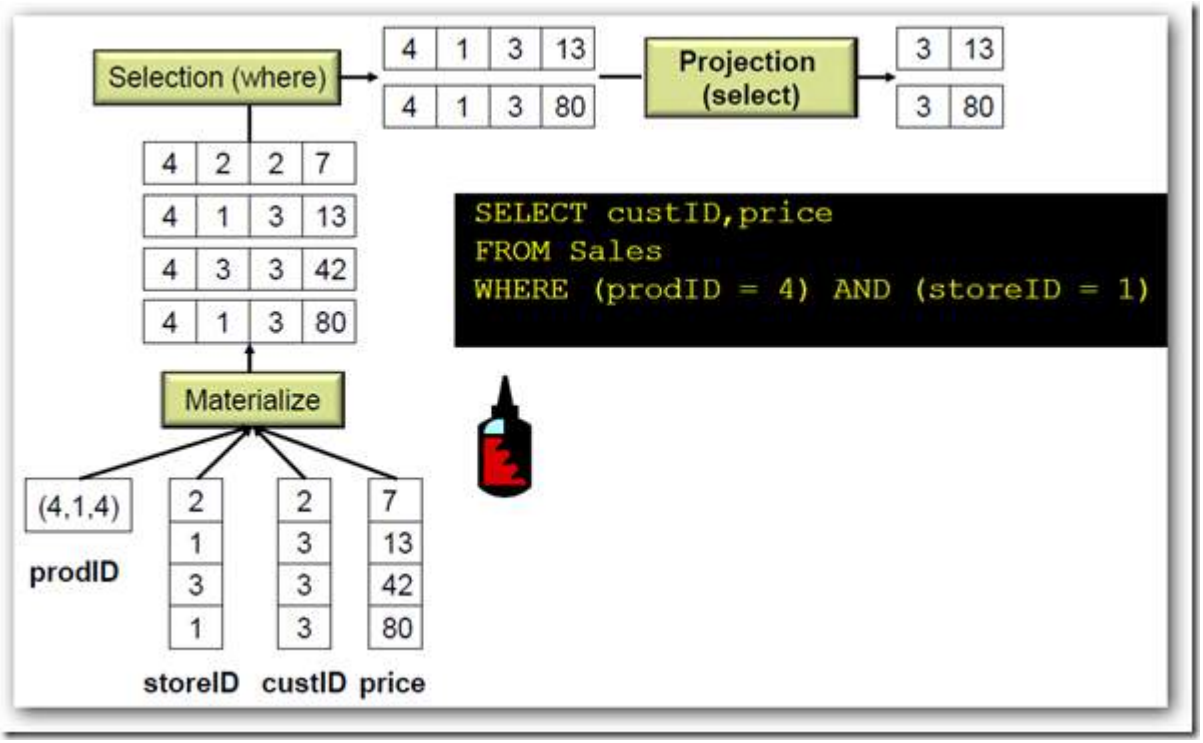
与每个operator之间只传递一个tuple的火山模型不同的是，向量化执行模型在不同operator之间传递的是cache-lined大小的块数据，可以理解为多组列向量。那么这将利于提高CPU效率和Cache的利用率。配合SIMD，CPU长流水线以及cacheline，将极大有利于提高吞吐量。

- 推迟物化

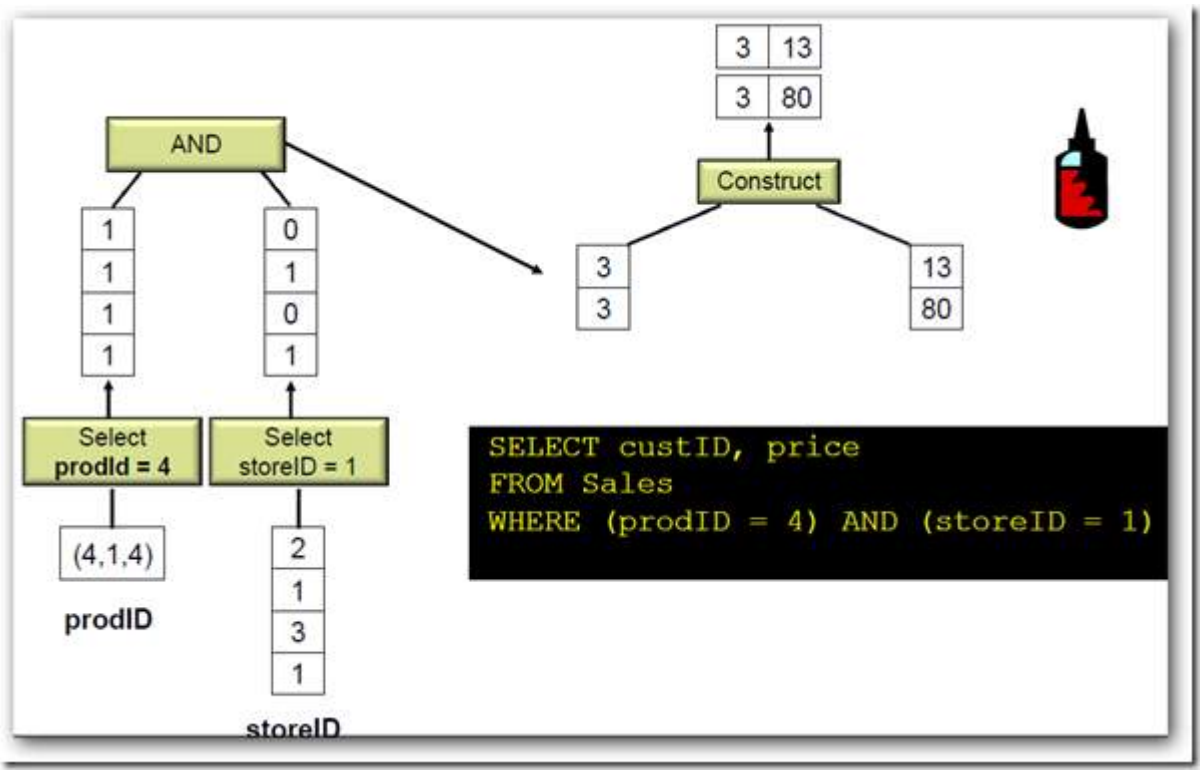
Late materialization or late tuple reconstruction ，指的是推迟把多列数据组织成更宽tuple的过程。

内部处理过程尽可能晚的去组合/物化最终的一整行数据，从而避免非必要列的构建

传统的行式数据库运算，运算一开始就会解压缩所有列的数据，然后组织成一个宽的tuple，再在这个宽tuple上做选择与投影。



而列式数据库，直到最后才解压还原数据，数据处理始终是以列为单位，这样有利于减少CPU，内存和网络传输消耗



1. 很多聚合与选择计算，压根不需要整行数据，过早物化会浪费严重；
2. 很多列是压缩过的，过早物化会导致提前解压缩，但很多操作可以直接下推到压缩数据上的；

3. 面向真正需要的列做计算，CPU的cache效率很高（100%），而行存因为非必要列占用了cache line中的空间，cache效率显然不高；
4. 针对定长的列做块迭代处理，可以当成一个数组来操作，可以利用CPU的很多优势（SIMD加速、cache line适配、CPU pipeline等）；相反，行存中列类型往往不一样，长度也不一样，还有大量不定长字段，难以加速；

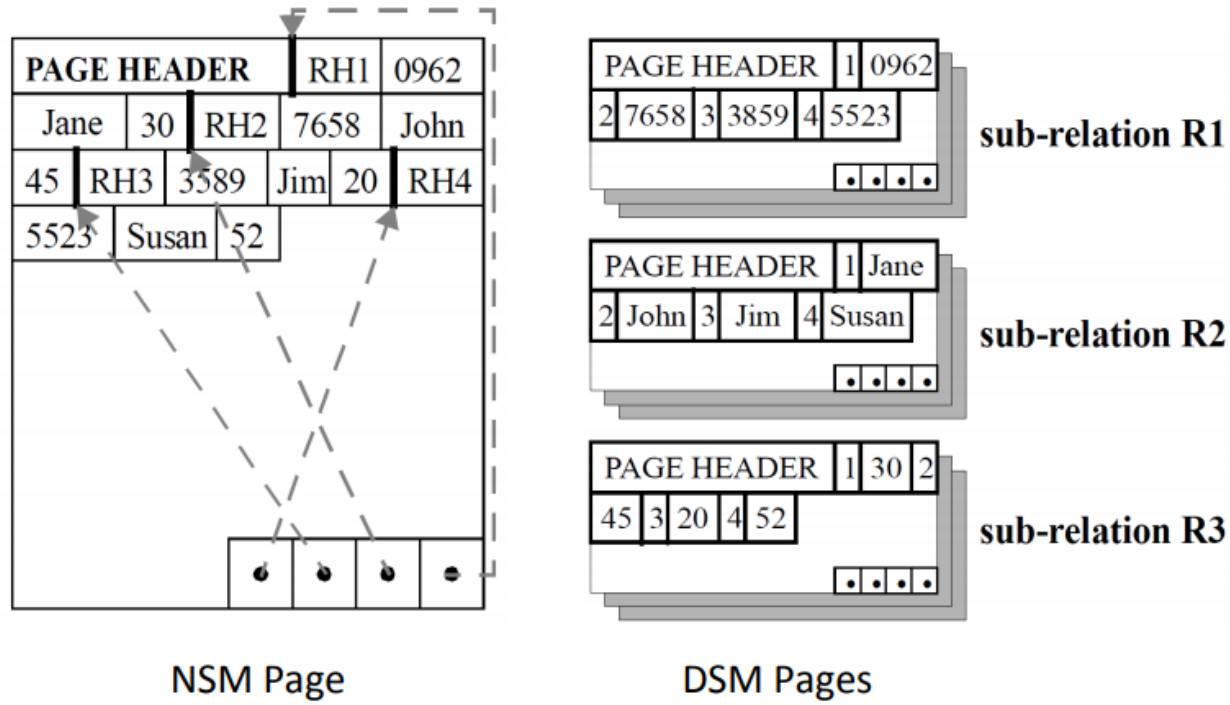
- 针对列来做压缩

因为同一列的数据往往数据类型相同，因此可以采用相同的压缩算法已达到更高的压缩比

- 更为高效的JOIN实现

HISTORY & TREND & PERF TRADEOFF

行式数据和列式数据分别在磁盘的一个Page的组织形式：



机械磁盘IO主要的瓶颈在于磁头的寻址过程，读写通常以一块block为单位。

相应的，数据库的Page对应着磁盘的一个block的大小，Page对应一个或者多个物理扇区，能够让数据库的Page和扇区对其，从而提高读写效率。

NSM： N-ary Storage Model

完整的行（即关系 relation）从 Header 开始依次存放。页的最后有一个索引，存放了页内各行的起始偏移量。由于每行长度不一定是固定的，索引可以帮助我们快速找到需要的行，而无需逐个扫描。

缺点：每次查询只涉及很小的一部分列，那多余的列依然要占用掉宝贵的内存以及 CPU Cache，从而导致更多的 IO.

DSM： Decomposition Storage Model

每个sub relation会存储单独一列的数据，同时页尾部也会存储索引。

如今，随着分布式文件系统的普及和磁盘性能的提高，**很多先进的 DBMS 已经抛弃了按页存储的模式**，但是其中的某些思想，例如**数据分区、分区内索引、行列混合**等，仍然处处可见于这些现代的系统中。

Column-Store Architectures

下面以C-Store对列式存储的体系结构做进一步讨论。

C-Store

- Read Optimized Store

每一列数据压缩后存储为单独的一个文件，并且可以按照规则按列排序。

CK的列式数据落盘就是这么设计的

- Write Optimized Store

新写入的数据，数据没有被压缩并且也没有垂直分区。后台会有线程周期性把WOS中的数据做排序，压缩并且落盘到ROS中。类比于CK的后台merge线程

- 每一列数据可能会根据不同的排序策略存储多次，每一组groups都可以叫做一个"projection"
- 每一列数据都会采取不同的压缩算法

压缩算法选取策略所要考虑的因素有：

1.column是否排好序

2.数据类型

3.列中的distinct values

- 稀疏索引加速查询

不是全量索引，而是存储了一列中每一个Page的第一个记录

- 采用非覆盖的Storage representation

update用delete+insert来替代

delete用额外存储一个delete列来标记这个tuple是否被删除

- C-Store被看为是Shared-Nothing

数据水平分片在不同节点上，（哈希/partition）

多个节点并行执行命令，得到的结果汇聚在一个节点上以供输出

列式存储实现细节

向量化执行

- 与tuple-at-a-time的火山模型相比，向量化执行模型在不同operator传递的是一个Vector，一个Vector可以理解为一列的许多行数据。通常行数的选择与L1 Cache大小有关，最好能刚好适配与Cache的大小

  向量化执行有什么好处呢?

  1.减少next这个虚函数的调用开销

2.更好的Cache局部性，vector的行数刚好能够fit如L1 Cache的大小

3.SIMD有了用武之地，同时把多行数据存入超大型寄存器

4.内存的访问也可以并行化

压缩

从信息论的角度来说，同列数据的信息熵会更低，单列数据的局部性也会更好，因此压缩效率会更好。

数据压缩后，花在IO的时间会更少，不仅能节省空间，而且SIMD能一次性加载更多的数据。

压缩算法：

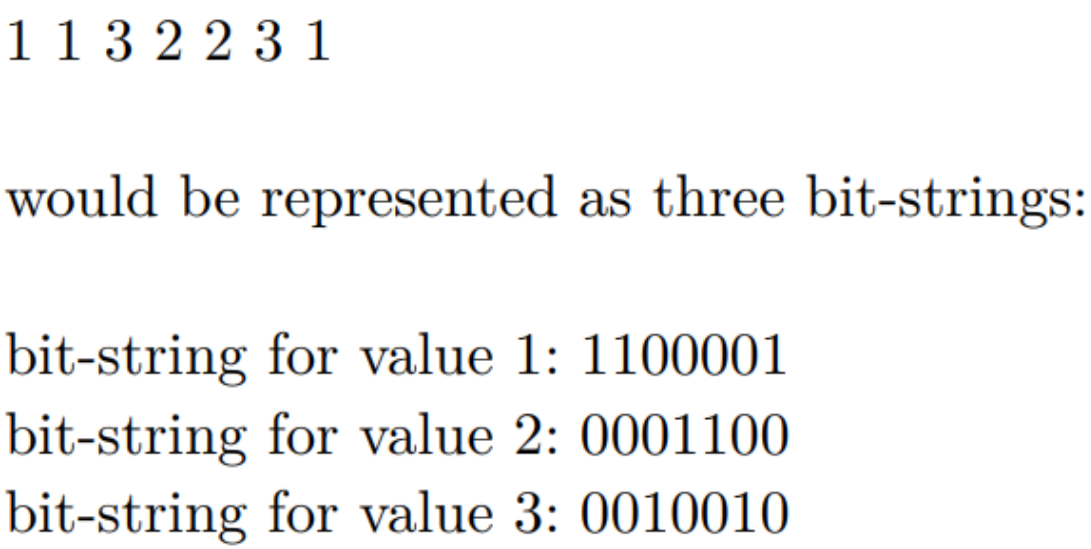
- Run-Length Encoding

eg: ('M', 1, 42)

如果某列的前42行数据都包含M，那么可以直接压缩成('M',1,42)

- Bit-Vector Encoding

适用场景：when columns have a limited number of possible data values



例如给定列中有以上数据，可以分别将其压缩成三个bit-string

位图索引：

ref https://www.cnblogs.com/lbser/p/3322630.html

位图索引适合只有几个固定值的列，如性别、婚姻状况、行政区等等,**即选择度或者基数较少**的几列

关键是缩小了存储空间，以使得内存遍历成为可能

- Dictionary字典编码

字典编码以列为单元进行操作，通过简单的转换将不同的值替换为不同的整数值(短整数优先)，将长文本值压缩为短整数值，因此并没有改变表的规模。通常情况下，企业数据的熵较低，即数据的重复度大，因此压缩效果较为理想。以性别列压缩为例：性别列仅包含两个值，若通过"m", "f"表示，那么需要1byte.假设全世界有70亿人口，那么需要70亿*1byte约为6.52GB. 如果使用字典压缩，1位足够表达相同信息，需要70亿1bit=0.81GB, 其中字典需要*21字节=2字节。压缩比例=未压缩大小/压缩大小约为8.

延迟物化

在列存数据库中，一个entiry的数据可能是存储在磁盘不同位置上。例如ck,每一列数据都是单独保存为一个文件。而行式数据库，一行的多列数据通常是compact地存储在一起的。

像ODBC和JDBC这种访问基本单元是entity-at-a-time（而不是column-at-a-time）也就是说面向行。

因此在执行query计划的时候，是需要这么一个过程：把多列数据整合成一行。因此对于列存数据库来说，这种Join-like的物化操作其实是非常常见的。

往往一个SQL中filter和project只会涉及到整行数据的部分列，但在传统行式存储下这些计算过程需要先将整行数据（因为整行数据是放在一起的）解析和提炼出来（虽然可以只选择与SQL有相关性的列参与后续计算，但整行解析的过程是少不了的），然后再做一些丢弃；而在列存中，每个列存储上都是独立的，因而真的可以做到只解析与SQL有相关性的列并过滤。

延迟物化的好处：

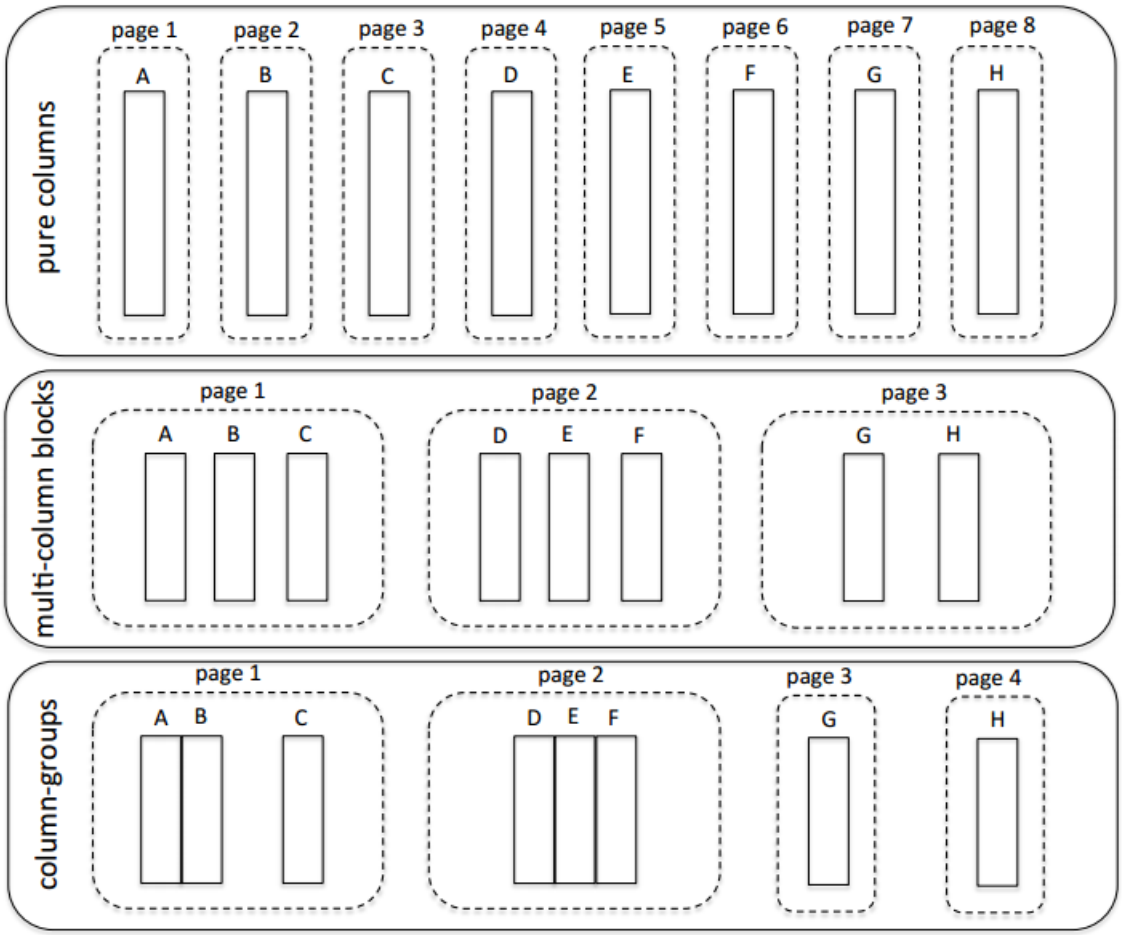
1.如selection和aggregation算子，在物化的时候可能会生成一些并不需要的tuples，那么延迟物化就是会尽可能减少这些中间tuples的生成。也就是说很多聚合与选择计算，压根不需要整行数据，过早物化会浪费严重

2.直接对列存数据操作，提高cache的利用率，避免加载不需要的列数据到cache中

3.很多列是压缩过的，过早物化会导致提前解压缩，但很多操作可以直接下推到压缩数据上的

4.针对定长的列做块迭代处理，可以当成一个数组来操作，可以利用CPU的很多优势（SIMD加速、cache line适配、CPU pipeline等）；相反，行存中列类型往往不一样，长度也不一样，还有大量不定长字段，难以加速

块组织方式



ref

https://zhuanlan.zhihu.com/p/35622907