**Deep Bodra**

**5801 1841**

**DBI Week 4(1/27 - 1/31)**

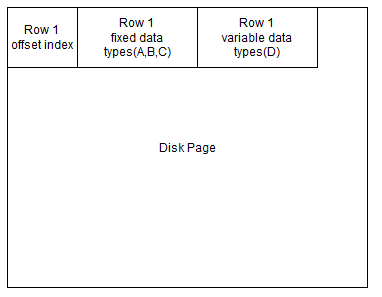
**Detailed explanation of the topics covered in class**

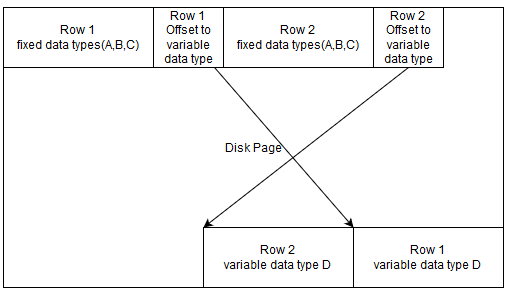
**1/28/2020**

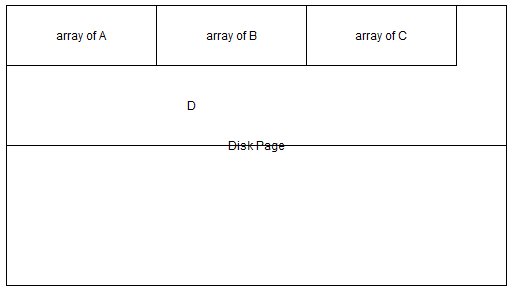
1. Introduction
   1. The mantra till 2010 was the “**Disk is slow, CPU is fast”**
   2. Gradually this changed and CPU’s need to be speeded up to cope up with the superfast speeds provided by M2 SSD’s
2. Row based databases
   1. These databases pack rows in the disk page
   2. Various types of strategies are possible for this as described below

Consider the relation R with 4 attributes A,B,C,D such that A,B and C are fixed data types whereas D is of variable type

* + 1. Strategy 1 (mixed representation)
       1. This stores the attributes of a tuple in the same order as when the relation was defined.
       2. So, the tuple1 will be stored as A1, B1, C1, D1 and then the remaining tuples if any in the similar manner
       3. A problem with this is that a lot of time is invested in parsing because we actually don’t know when the value for variable data type ends
    2. Strategy 2 (separate representation)
       1. This separates the fixed and variable data types



* + - 1. The markers for the variable data types are stored in the offset index
      2. This gets rid of the parsing time of Strategy 1
    1. Strategy 3
       1. 
       2. This strategy fills fixed data types into the disk from top to bottom and the variable data types from bottom to top
       3. After the fixed data type values of a row it stores the offset of the corresponding variable data type value (D)
       4. Drawback
          1. We read more attributes even if we are only worried about a few
          2. The cache behaviour is bad
       5. A possible solution is to use **weaved relation**



Fixed data types are stored as their arrays and then the variable data types

* 1. Column Oriented
     1. The data is stored column wise in the disk
     2. Advantages
        1. Reduced Disk Traffic
        2. Good Cache Line Utilisation
        3. Good Prefetching
     3. The reads and inserts are fast
     4. Updates and deletes are slow
     5. Strategy to delete: mark as delete and delete when explicitly requested (eg. vacuum() function in Postgres)
  2. CSTORE
     1. It is a column oriented DB (by Brown University, MIT, Brandeis University)
     2. It was 100 times faster than other databases at that time but did not take advantage of instructions like AVX, SSE, etc
  3. Skinny relations
     1. A Hack to get Column store DB from row store
     2. Companies like Postgres, MySQL used this to keep up with the growing trend of Column store
     3. Each attribute of a relation is now a table with 2 attributes (row\_id, value of that attribute)
     4. This did not give performance anywhere near to Column store
  4. How to get fast linear scan?
     1. No file system. With file system you can get a maximum of 5 MB/s
     2. Large number of independent requests in the request queue
  5. DataPath
     1. Launched in 2010
     2. Variant 1
        1. 70 Drives (50-100MB/s) | 3 RAID controller
        2. The maximum throughput back then was 700 MB/s but datapath yielded 1.86GB/s
     3. Variant 2
        1. 16SSD | 1 RAID controller | using Just a Bunch Of Drives
        2. Throughput=4.3GB/s
     4. The disk is divided into chunks with each chunk containing 2 million tuples stored (column wise)
     5. Each relation is a collection of these chunks

**1/30/2020**

1. Problems with INT representation
   1. Floats have a standard IEEE representation so their representation in memory does not introduce discrepancy between different machines
   2. But, INT’s have Little ENDIAN (in Intel) and Big ENDIAN (in Power PC)
2. Compression for DISK representation
   1. We can even compress the data stored in the disk page
   2. The compression usually makes use of the dictionary that also needs to be stored on the disk
   3. If the page size is 4KB then it’s not worth compressing as the size of the dictionary will be as close to the actual data
   4. Also the compression ratio would be low
3. Process Compressed Columns
   1. Compressed columns may speed up the query execution time
   2. For eg. if we have a column as 1,1,1 then the compressed version will be (value, count) = (1,3) and the sum of the column can be computed easily as 1\*3
   3. Sorting the columns will help more
   4. But this proves to be efficient only for simple analytical queries
   5. The efficiency drops for complex analysis
4. Datapath continued
   1. Adaptive Execution
      1. Compressed when CPU is available
      2. Uncompressed otherwise
   2. Joins
      1. For joins the column representation is converted to row representation
   3. Query Rewriting
      1. The query is transformed into a query which takes less time to execute

--THANK YOU--