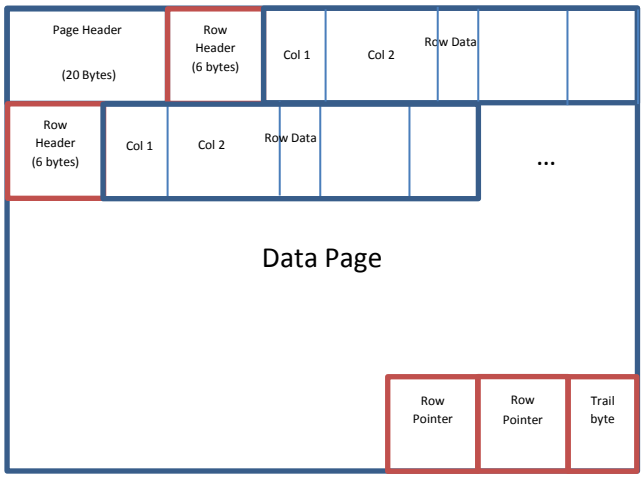
Row or Columnar Database

If someone is evaluating database or data stores to use in their application, there are so many options to choose from especially in the data ware house space. If narrowed down to the relational database (RDBMS) paradigm, one of the choices to make is whether to use row based or columnar based database. Vendors claim superiority of one over the other on whether their product is columnar or row based. So we looked into the details about columnar and row databases to understand the fundamental differences. The following is the summary.

Why the name?

Row Based RDBMS (R-RDBMS):

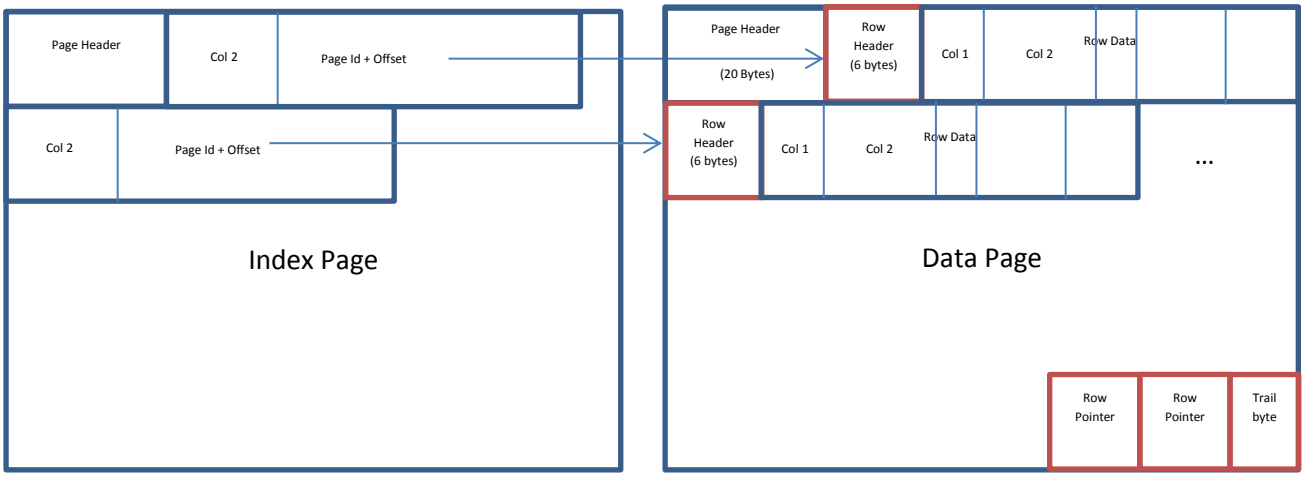
In a row based DBMS, data related to a tuple (row) i.e. all the column data are stored contiguously on disk. For IO efficiency, disk reads and writes are done at block size, for e.g. a 4K (4096 byte) block size by Operating Systems. Database management systems use "pages" of size which is a multiple of the block size to read and write data to disk. In R-RDBMS rows of data are stored in data pages and if the row size is less than half the page size, then multiple rows are stored in a single page. When a row is required by a query, the whole page in which the row is stored is retrieved back from the disk into the memory for further processing. The following is a representative page layout based on one of the RDBMS used in the industry.



As we can see the R-RDBMS stores additional information regarding each page and rows with in the page to help with maintaining the ACID (Atomicity, Consistency, Isolation, Durability) property expected out of RDBMS.

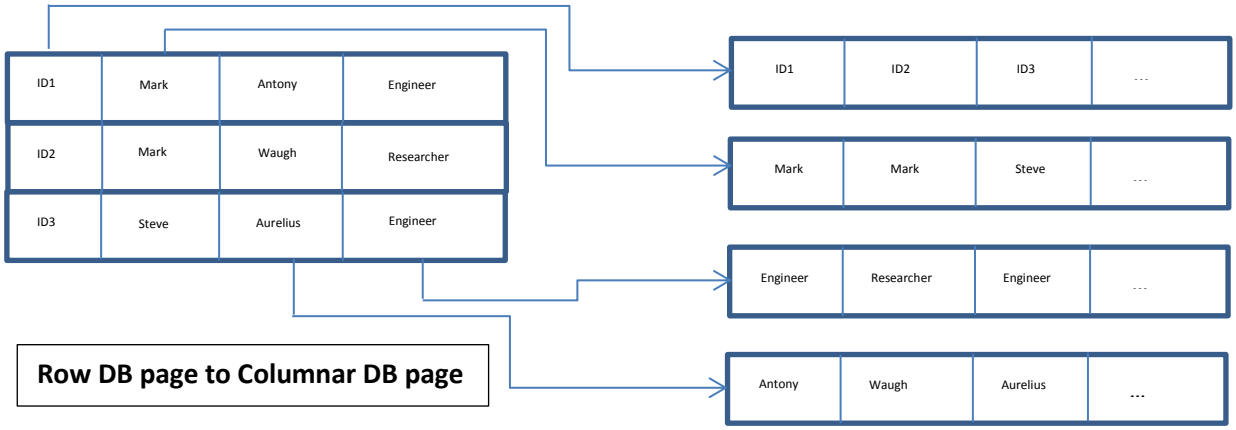
In order to improve query performance, R-RDBMS uses additional structure called indexes. Indexes store the indexed column value, the page in which the row with the indexed column value is stored on disk and the offset within the page to reach out to the particular row. If an index is not present, when a query is executed against a table, the DBMS needs to read through all the pages from the disk pertaining to the table to find the rows which satisfy the query. If the index is present and the query uses the indexed columns in its predicate, the DBMS can use the index to identify the rows which satisfy the query and read the pages where the rows are stored reducing the time to identify the rows. This also reduces the amount of data read from the disk i.e.

reducing the slowest operation in the query processing sequence which is disk IO. The following is a representative index page in a typical DBMS. There can be other representations based on various index structures like BTree, Bit map etc.



Columnar RDBMS (C-RDBMS):

As you may have guessed, columnar databases store each column data from all the tuples together. The following diagram shows the translation of data storage between a row based DBMS and columnar DBMS.



Contrary to storing all the column data corresponding to a row sequentially in a page, values for each column in rows are stored together in the same page. This results in data for each row getting stored in different pages. When a query requires data for a row, column data for the row is pulled from all the pages storing the column values, appends them together before returning it to the user as a single row. The sequence in which the column values are stored in the database pages determines the row to which corresponds to. For e.g. the second entry stored in the various pages "ID2", "Mark", "Waugh", "Researcher" corresponds to the same row which is row 2.

By storing the columns separately, each column acts as an index since the sequence of storage identifies the row. For e.g. if a query requests the row with first name 'Steve' the DBMS can identify the row number using

the pages storing the column "First Name" which in this case is row 3. Then the DBMS can retrieve the third entry from pages storing all the other column values stitch them together and return the row back to the user.

How they differ?

Given the understanding of the key difference between R and C-RDBMS, we can look at how they differ operationally.

- If the usage pattern involves retrieval and update of all or most of the columns in a row like in an OLTP application, then R-RDBMS is a better option than the C-RDBMS. The reason being that the C-RDBMS needs to retrieve the columns values separately and stitch them together to return the row as a response and this doesn't provide the performance expected in an OLTP environment. Also in C-RDBMS updates need to be made on multiple pages in contrast to updating a single page in R-RDBMS which is inefficient. C-RDBMS is primarily suited for data ware housing where the usage pattern is read only.
- If the usage pattern involves retrieval of all the columns in a row in bulk, then R-RDBMS is a better option due to the same reason described above. But if the bulk retrieval involves only a small subset of columns, then the C-RDBMS will perform better. The reason being that C-RDBMS can deal with the subset of columns since they are stored separately while R-RDBMS need to bring in all the rows and columns into memory from the disk and process it through CPU to eliminate the unwanted columns. Some R-RDBMS products like Netezza may be able to eliminate the unwanted columns using specialized hardware during disk read but still need to deal with all the rows columns.
- If the usage pattern involves aggregation on columns then C-RDBMS performs better than the R-RDBMS since they can act on individual columns efficiently compared to R-RDBMS.
- C-RDBMS can implement optimization techniques like late materialization where conditions on columns can be applied separately, identify the rows which satisfies all the conditions before retrieving the columns to generate rows whereas R-RDBMS needs to retrieve rows much earlier to identify the satisfying rows and to return to the user.
- Storage required for C-RDBMS will be less than the R-RDBMS since they don't have the same page and row overheads as R-RDBMS. Also they don't need additional structures like indexes since the columns themselves act as indexes.
- Compression on data is efficient on C-RDBMS since data which are similar are stored together compared to R-RDBMS where mixed data in rows are stored together. This helps reduce space usage in C-RDBMS and also improves the disk IO since the data is much compressed.

Can R-RDBMS implement C-RDBMS?

One can try to mimic C-RDBMS storage in an R-RDBMS using any of the following techniques

- Store columns as separate tables with a common identifier column to identify the row to which the columns value corresponds to.

- Create indexes for each of the columns in a table so that queries can be satisfied by using the indexes only.

Also there are commercial DBMS products which support both columnar and row based storage. Apart from the increased (more than double) storage requirement to implement these techniques, research from MIT database group shows that these techniques do not provide the same performance as the C-RDBMS for all the usage patterns for which C-RDBMS is best suited for.

Summary

C-RDBMS are more suited for data warehousing use cases and it is how they are utilized currently in the industry. Also C-RDBMS may perform much better when usage involves small set of column retrieval and column aggregations. R-RDBMS are good for use cases where data is dealt at the row level and where updates are often made on rows. C-RDBMS and R-RDBMS vendors may find ways to incorporate some of the advantages of the other in their product. The key is to understand the data usage pattern and choose the best product which matches the usage. Even though we have eliminated other complexities in a typical DBMS system and looked only at the fundamental difference between R and C RDBMS, hope it helps you choose the best option for your application.

 **bnair@asquareb.com**

 **blog.asquareb.com**

 **https://github.com/bijugs**

 **@gsbiju**

 **http://www.slideshare.net/bijugs**