

# A Vision towards Column-Oriented Databases

Punam Bajaj      Simranjit Kaur Dhindsa  
 Department of Computer Science & Engineering  
 Chandigarh Engg. College, Landran, Mohali, India

## Abstract:

*This paper describes a basic difference between column-oriented databases and traditional row-oriented databases. As applications require higher storage and easier availability of data, the demands are satisfied by better and faster techniques [1]. Column-oriented database systems (Column-stores) have attracted a lot of attention in the past few years. A column-oriented DBMS is a database management system (DBMS) that stores its content by column rather than by row as in row-oriented databases. This has advantages for data warehouses and library catalogues where aggregates are computed over large numbers of similar data items [4]. In this paper, we discuss how Column oriented Database better than traditional row-oriented DBMSs. This paper focuses on conveying an understanding of columnar databases and the proper utilization of columnar databases within the enterprise.*

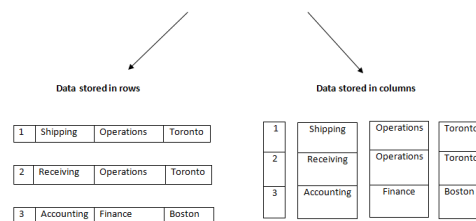
**Keywords:** *Column–Stores, Column–oriented DBMS, Data WareHouses, Columnar Database.*

## I Introduction:

Faced with massive data sets, a growing user population, and performance-driven service level agreements, organizations everywhere are under extreme pressure to deliver analyses faster and to more people than ever before. That means businesses need faster data warehouse performance to support rapid business decisions,

added applications, and better system utilization. And as data volumes continue to increase driven by everything from longer detailed histories to the need to accommodate big data companies require a solution that allows their data warehouse to run more applications and to be more responsive to changing business environments. Plus, they need a simple, self-managing system that boosts performance but helps reduce administrative complexities and expenses. Column Oriented DBMS provides unlimited scalability, high availability and self-managing administration [5].

Employee_ID	Job	Dept	City
1	Shipping	Operations	Toronto
2	Receiving	Operations	Toronto
3	Accounting	Finance	Boston



**Fig 1. Base concept**

In fig 1, Starting with a generic table, There are two obvious ways to map database tables onto a one dimensional interface: store the table row-by-row or store the table column-by-column. The row-by-row approach keeps all information about an entity together. In the example above, it will store all information about the first employee, and then all information about the second employee, etc. The column-by-column approach keeps all attribute information together: all of the employee id's will be stored consecutively, then all of the employee job, etc. Both approaches are reasonable designs and typically a choice is made based on performance

expectations. If the expected workload tends to access data on the granularity of an entity (e.g., find an employee, add an employee, delete an employee), then the row-by-row storage is preferable since all of the needed information will be stored together. On the other hand, if the expected workload tends to read per query only a few attributes from many records (e.g., a query that finds the most common e-mail address domain), then column-by-column storage is preferable since irrelevant attributes for a particular query do not have to be accessed [2].

Traditionally, Row oriented databases are better suited for transactional environments, such as a call center where a customer's entire record is required when their profile is retrieved.

Column-oriented databases are better suited for analytics, where only portions of each record are required. By grouping the data together like this, the database only needs to retrieve columns that are relevant to the query, greatly reducing the overall I/O needed [6].

## II Core difference of Columnar Database than row-oriented Database:

The world of relational database systems is a two-dimensional world. Data is stored in tabular data structures where rows correspond to distinct real-world entities or relationships, and columns are attributes of those entities. There is, however, a distinction between the conceptual and physical properties of database tables. This aforementioned two-dimensional property exists only at the conceptual level. At a physical level, database tables need to be mapped onto one dimensional structure before being stored. This is because common computer storage media (e.g. magnetic disks or RAM), despite ostensibly being multi-dimensional, provide only a one dimensional interface. For example, a database might have this table [2].

EmpId	Lastname	Firstname	Salary
1	Wilson	Joe	40000
2	Yaina	Mary	50000
3	John	Cathy	44000

Fig 2. Two Dimensional Table

This simple table includes an employee identifier (EmpId), name fields (Lastname and Firstname) and a Salary. The database must coax its two-dimensional table into a one-dimensional series of bytes, for the operating system to write it to either the RAM, or hard drive, or both. A row-oriented database serializes all of the values in a row together, then the values in the next row, and so on.

1, Wilson, Joe, 40000;  
2, Yaina, Mary, 50000;  
3, John, Cathy, 44000;

A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on.

1, 2, 3;  
Wilson, Yaina, Johnson;  
Joe, Mary, Cathy;  
40000, 50000, 44000;

This is a simplification. Partitioning, indexing, caching, views, OLAP cubes, and transactional systems such as write ahead logging or multiversion concurrency control all dramatically affect the physical organization [4].

## III Limitations of Row oriented DBMS's:

Historically, database system implementations and research have focused on the row-by-row data layout, since it performs best on the most common application for database systems: business transactional data processing. However, there are a set of emerging applications for database systems for which the row-by-row layout performs poorly. These applications are more analytical in nature, whose goal is to read

through the data to gain new insight and use it to drive decision making and planning. The nature of the queries to data warehouses (analytical databases) is different from the queries to transactional databases. Queries tend to be:

- 1) **Less Predictable:** In the transactional world, since databases are used to automate business tasks, queries tend to be initiated by a specific set of predefined actions. As a result, the basic structure of the queries used to implement these predefined actions is coded in advance, with variables filled in at run-time. In contrast, queries in the data warehouse tend to be more exploratory in nature. They can be initiated by analysts who create queries in an ad-hoc, iterative fashion.
- 2) **Longer Lasting:** Transactional queries tend to be short, simple queries (“add a customer”, “find a balance”). In contrast, data warehouse queries, since they are more analytical in nature, tend to have to read more data to yield information about data in aggregate rather than individual records.
- 3) **More Read-Oriented Than Write-Oriented:** Analysis is naturally a read-oriented endeavor. Typically data is written to the data warehouse in batches, followed by many read only queries. Occasionally data will be temporarily written for “what-if” analyses, but on the whole, most queries will be read-only.
- 4) **Attribute-Focused Rather Than Entity-Focused:** Data warehouse queries typically do not query individual entities; rather they tend to read multiple entities and summarize or aggregate them. Further, they tend to focus on only a few attributes at a time rather than all attributes.

As a consequence of these query characteristics, storing data row-by-row is no longer the obvious choice; in fact, specially as a result of the latter two characteristics, the column-by-column storage layout can be better [2].

#### IV Evolution of Column Oriented DBMSs:

The following are some cited advantages of column-stores:

Improved bandwidth utilization: In a column-store, only those attributes that are accessed by a query need to be read off disk (or from memory into cache). In a row-store, surrounding attributes also need to be read since an attribute is generally smaller than the smallest granularity in which data can be accessed.

Improved data compression: Storing data from the same attribute domain together increases locality and thus data compression ratio (especially if the attribute is sorted). Bandwidth requirements are further reduced when transferring compressed data.

Improved code pipelining: Attribute data can be iterated through directly without indirection through a tuple interface. This results in high IPC (instructions per cycle) efficiency, and code that can take advantage of the super-scalar properties of modern CPUs.

Improved cache locality: A cache line also tends to be larger than a tuple attribute, so cache lines may contain irrelevant surrounding attributes in a row-store. This wastes space in the cache and reduces hit rates [3,4].

#### Additional Considerations:

In addition to better performance, the column-orientation aspect of column-based database supplies a number of useful benefits to those wishing to deploy fast business intelligence databases.

First, there is no need for indexing as with traditional row-based databases. The elimination of indexing means: (1) less overall storage is consumed in columnar databases because indexes in legacy RDBMS's often balloon the storage cost of a database to double or more the initial data size; (2) data load speed is increased because no indexes need to be maintained; (3) ad-hoc DML work speed is increased because no index updates are performed; (4) no indexing design or tuning work is imposed on the database IT staff.

Second, there is far less design work forced on database architects when a column-based database is used. The need for complicated partitioning schemes, materialized view or summary table designs, and other such work is completely removed because column databases need none of these components to achieve superior query performance.

### **Data Compression:**

Compression is a technique used by many DBMSs to increase performance. Compression improves performance by reducing the size of data on disk, decreasing seek times, increasing the data transfer rate and increasing buffer pool hit rate [7]. One of the most-often cited advantages of Column-Stores is data compression. Intuitively, data stored in columns is more compressible than data stored in rows. Compression algorithms perform better on data with low information entropy (high data value locality) [1]. Imagine a database table containing information about customers (name, phone number, e-mail address, e-mail address, etc.). Storing data in columns allows all of the names to be stored together, all of the phone numbers together, etc. Certainly phone numbers will be more similar to each other than surrounding text fields like e-mail addresses or names. Further, if the data is sorted by one of the columns, that column will be super-compressible. Column

data is of uniform type; therefore, there are some opportunities for storage size optimizations available in column-oriented data that are not available in row-oriented data. Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth. Infobright is an example of an open source column oriented database built for high-speed reporting and analytical queries, especially against large volumes of data. Data that required 450GB of storage using SQL Server required only 10GB with Infobright, due to Infobright's massive compression and the elimination of all indexes. Using Infobright, overall compression ratio seen in the field is 10:1. Some customers have seen results of 40:1 and higher. Eg. 1TB of raw data compressed 10 to 1 would only require 100 GB of disk capacity [6].

### **V Conclusion:**

Column oriented DBMS is an enhanced approach to service the needs of Business Intelligence (BI), data warehouse, and analytical applications where scalability, performance and simplicity are paramount. It delivers a future-proof data management infrastructure with the ability to scale from a single database node to a multi-node. When you need to analyse a mountain of data, there simply is no substitute for column database technology that ensure scalable, linear performance capabilities and to deliver faster performance than legacy databases that use all of them just to cross the finish line second. The Columnar Database is evolving software which can overcome the lacks of the scope of row oriented databases. It provides a range of benefits to an environment needing to expand the envelope to improve performance of the overall analytic workload. It is usually not difficult to find important workloads that are column selective, and therefore benefit tremendously from a columnar orientation.

## VI Future Work:

Columnar database benefits are enhanced with larger amounts of data, large scans and I/O bound queries. While providing performance benefits, they also have unique abilities to compress their data. Therefore, Columnar can now be used in a data mart or a large integrated data warehouse[5]. Oracle describes the Exadata columnar compression scheme where Hybrid Columnar Compression on Exadata enables the highest levels of data compression and provides enterprises with tremendous cost-savings and performance improvements due to reduced I/O. Average storage savings can range from 10x to 15x depending on which Exadata Hybrid Columnar Compression feature is implemented; customer benchmarks have resulted in storage savings of up to 204x! Exadata Hybrid Columnar Compression is an enabling technology for two new Oracle Exadata Storage Server features: Warehouse Compression and Archive Compression. We can explore Exadata Hybrid Columnar Compression – the next generation in compression technology [8].

## VII References:

- [1] Daniel J. Abadi, Peter A. Boncz, Stavros Harizopoulos, **Column-oriented Database Systems**, *VLDB '09*, August 24-28, 2009, Lyon, France.
- [2] Daniel J. Abadi, **Query Execution in Column-Oriented Database Systems**,
- [3] D. J. Abadi, S. R. Madden, N. Hachem, **Column-stores vs. row-stores: how different are they really?**, in: SIGMOD'08, 2008, pp. 967–980.
- [4] Column-Oriented DBMS, wikipedia
- [5] TeraData Columnar, www.TeraData.com
- [6] Infobright, **Analytic Applications With PHP and a Columnar Database**(2010), 403-47 Colborne St Toronto, Ontario M5E 1P8 Canada.

[7] Miguel C. Ferreira, **Compression and Query Execution within Column Oriented Databases.**

[8] Oracle Exadata, **Hybrid Columnar Compression (HCC) on Exadata.**