A row store is a collection of records that contain the fields of a single row in a table. The entries of a column are kept in contiguous memory regions in a column store. Column-based storage is often appropriate for large tables with frequent changes. Row tables, on the other hand, perform better in terms of update and insert performance. Information about a logical entity (e.g., a person) is saved in numerous locations on disk in a column-store (e.g., name, e-mail address, phone number, etc. are all stored in distinct columns), whereas in a row store such information is normally co-located in a single row of a table. Under the hood, column-oriented databases store all values from each column together, whereas row-oriented databases store all entries in a row together. Consider how the values of each column are saved in distinct files to help visualize this.

Prior research has revealed that essential column-oriented DBMS improvements include: Late materialization (also known as vectorized query processing when paired with the block iteration optimization described below), in which columns received from disk are put together into rows as late as practicable in a query plan. Instead of employing Volcano-style per-tuple iterators, block iteration involves passing several values from a column as a block from one operator to the next. If the values have a fixed width, they are iterated as an array.

Queries over data warehouses, especially those modeled using a star schema, frequently contain the following structure: Use selection predicates on one (or more) dimension tables to limit the set of tuples in the fact table. Then, on the constrained fact table, execute some aggregation, frequently grouping by other dimension table features. As a result, joins between the fact and dimension tables must be performed for each selection predicate and aggregate grouping.

On the data warehousing benchmark, SSBM, this research evaluated the performance of C-Store to many variations of a commercial row-store system. The authors demonstrated that attempting to imitate the physical architecture of a column-store in a row-store using approaches such as vertical partitioning and index-only planning does not result in satisfactory performance. This latency is attributed to high tuple reconstruction costs as well as significant per-tuple overheads in narrow, vertically partitioned databases. They dissected why a column-store can handle column-oriented data so well, discovering that late materialization increases performance by a factor of three and compression improves speed by a factor of two on average, or by an order of magnitude on queries that access sorted data.