**Comparison of Row-Stores and Column-Stores Based on the Given Aspects**

**1. Usability for Online Analytical Processing (OLAP)**

* **Row-Stores**
  + Historically used for **Online Transaction Processing (OLTP)**.
  + Data is stored **tuple-by-tuple**, meaning entire rows are retrieved during queries.
  + **Not optimal** for OLAP queries that focus on **aggregations and summaries**.
  + Queries that **scan large amounts of data** suffer from **high I/O overhead** since unnecessary columns are read.
* **Column-Stores**
  + Designed specifically for **OLAP workloads**.
  + Data is stored **column-by-column**, allowing efficient reading of only the required columns.
  + **Aggregation functions (SUM, AVG, COUNT)** perform better because they operate on a **single column** without scanning entire rows.
  + Reduces **disk I/O** significantly, leading to **faster query execution**.

✅ **Conclusion**: Column-stores are **better suited** for OLAP because they enable faster aggregations and reduce the data read from disk.

**2. Compression Techniques**

* **Row-Stores**
  + Compression is applied at the **relation or partition level**.
  + Uses **generic compression techniques** (e.g., **Lempel-Ziv, Huffman Coding**).
  + Achieves **lower compression ratios** (typically **2:1 to 5:1**) due to mixed data types in a row.
  + Since tuples contain different data types, choosing a **universal compression strategy** is difficult.
* **Column-Stores**
  + **Highly efficient compression** due to **homogeneous column data types**.
  + Uses **Run-Length Encoding (RLE)**, **Dictionary Encoding**, **Bit-Packing**, and **Delta Encoding**.
  + Achieves **high compression ratios** (typically **10:1 to 40:1**).
  + Less **I/O overhead** due to compressed storage.
  + Can **process queries directly on compressed data** without full decompression.

✅ **Conclusion**: Column-stores **compress data more efficiently** and reduce I/O overhead compared to row-stores.

**3. Query Execution**

* **Row-Stores**
  + Uses a **tuple-oriented execution model**.
  + Queries must **scan entire rows**, leading to **high disk access**.
  + **Joins are expensive** because entire tuples are loaded into memory.
  + Queries are **less efficient** for **aggregation-heavy** workloads.
  + **Materialization occurs early**, leading to **higher memory usage**.
* **Column-Stores**
  + Uses a **vectorized execution model** (operates on columns instead of rows).
  + **Faster query processing** because only the required columns are scanned.
  + **Joins are more efficient** due to column-based storage.
  + **Late materialization** reduces memory usage by delaying tuple reconstruction.
  + Can **execute queries on compressed data**, reducing decompression overhead.

✅ **Conclusion**: Column-stores provide **better query execution performance** for **analytic queries** due to vectorized execution and late materialization.