**Parquet Data Format**

Key requirements and characteristics include:

- **Columnar Storage**: Data is stored by columns rather than rows. This allows query engines to read only the specific columns needed for a query (column pruning), significantly reducing I/O operations and processing time, especially on large datasets.

- **Efficient Compression and Encoding**: Grouping data of the same type in columns enables highly effective compression algorithms (like Snappy, Gzip, Zstandard). This results in significantly smaller file sizes (often 75% smaller than CSVs) and lower storage costs.

- **Optimized Query Performance**: Parquet leverages metadata (including minimum/maximum values within column chunks) to support "predicate pushdown". This allows the query engine to skip entire blocks of data that don't match the filter criteria, drastically improving performance for analytical and aggregation queries.

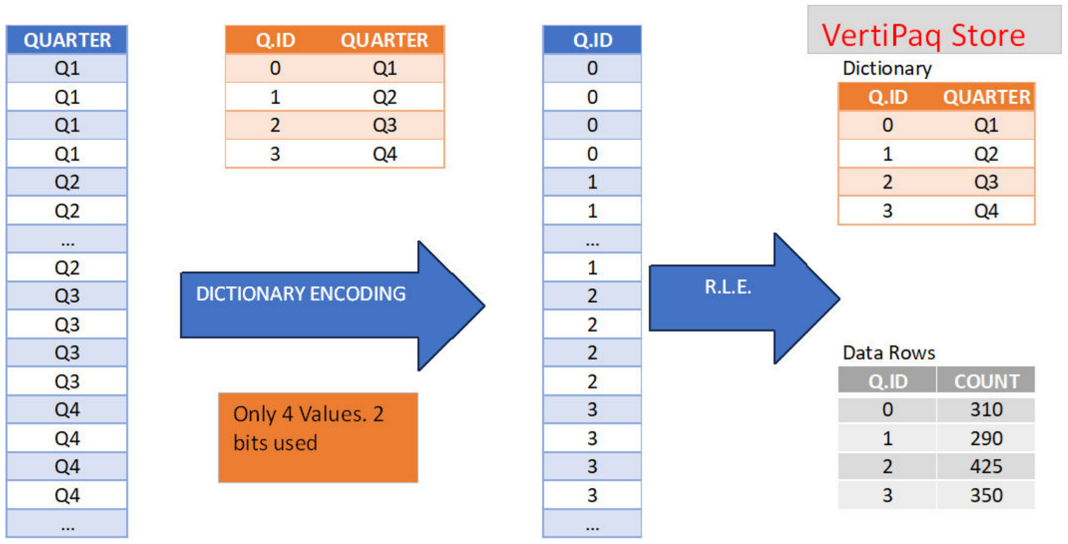
- **Schema Evolution**: The format supports schema evolution, allowing users to add new columns or modify schemas without rewriting the entire dataset, which provides flexibility in dynamic data environments.

- **Language Agnostic & Interoperability**: As an open-source format, Parquet is language-agnostic and widely supported across various data processing frameworks and tools (e.g., Apache Spark, Hive, Python, Java, Amazon Athena, Google BigQuery).

- **Support for Complex Data Types**: It natively handles complex and nested data structures (like arrays, maps, and structs), which is a challenge for simpler formats like CSV or JSON.

- **Binary Format**: Parquet stores data in a binary format, making it machine-optimized and efficient for processing, though not human-readable.

- **Splittable**: Parquet files can be split into independent row groups, enabling parallel processing in distributed computing frameworks like Apache Hadoop.



| QUARTER |
|---|
| Q1 |
| Q1 |
| Q1 |
| Q1 |
| Q2 |
| Q2 |
| ... |
| Q2 |
| Q3 |
| Q3 |
| Q3 |
| Q3 |
| Q4 |
| Q4 |
| Q4 |
| Q4 |
| ... |

| Q.ID | QUARTER |
|---|---|
| 0 | Q1 |
| 1 | Q2 |
| 2 | Q3 |
| 3 | Q4 |

| Q.ID |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |
| ... |
| 1 |
| 2 |
| 2 |
| 2 |
| 2 |
| 3 |
| 3 |
| 3 |
| 3 |
| ... |

Dictionary

| Q.ID | QUARTER |
|---|---|
| 0 | Q1 |
| 1 | Q2 |
| 2 | Q3 |
| 3 | Q4 |

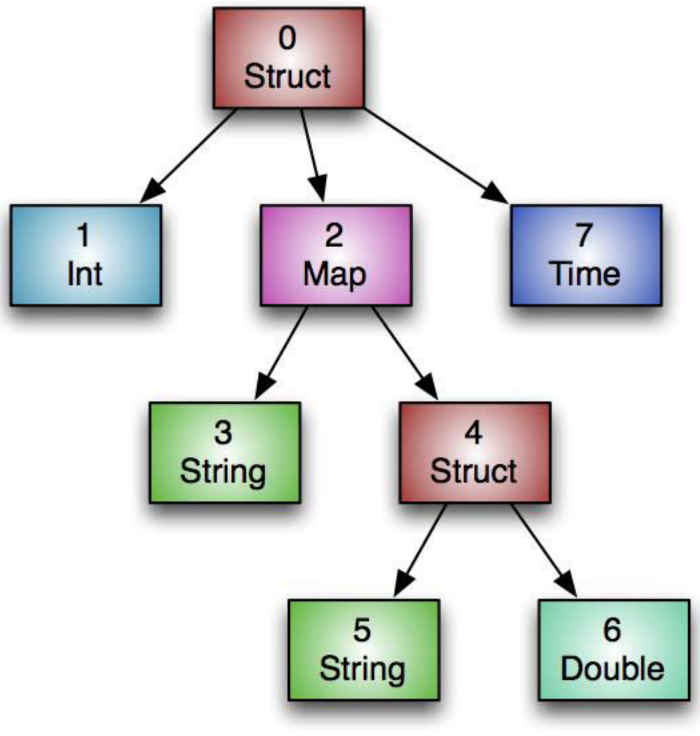
Data Rows

| Q.ID | COUNT |
|---|---|
| 0 | 310 |
| 1 | 290 |
| 2 | 425 |
| 3 | 350 |

**Optimized Row Columnar (ORC)**

Requirements and Characteristics:

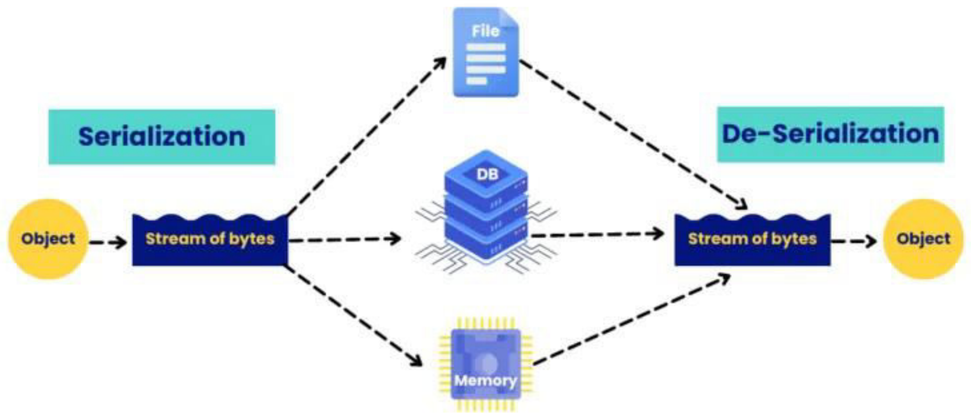
- **Columnar Storage:** Data is organized by columns rather than rows. This allows queries to read, decompress, and process only the necessary columns, which drastically reduces I/O and improves performance for analytical queries.
- **Efficient Compression:** ORC uses type-specific and general-purpose compression (like ZLIB, Snappy, or Zstd) on data within columns, which tend to have similar values. This results in a significantly smaller storage footprint compared to row-based formats like CSV or Avro.
- **Lightweight Indexing & Statistics:** ORC files contain built-in indexes and statistics (min/max values, counts, sum, null values) at both the file and stripe levels. This enables predicate pushdown, allowing query engines to skip entire groups of rows (called "stripes", typically 64-250MB in size) that do not match the query's filter criteria.
- **Schema Evolution:** The format supports schema changes, such as adding or removing columns, without requiring a rewrite of all existing data.
- **Support for Complex Types:** ORC accommodates a wide range of data types, including primitive types (int, float, string, date, etc.) and complex types (structs, lists, maps, and unions).
- **ACID Transaction Support:** ORC provides support for Atomicity, Consistency, Isolation, and Durability (ACID) properties, making it suitable for data warehousing and transactional workloads in systems like Apache Hive.



**AVRO**

Requirements and Characteristics:

- **Big Data Processing:** Used as an input/output format in frameworks like Apache Hadoop, Spark, and Flink.
- **Real-Time Streaming:** Integrated with systems like Apache Kafka for efficient serialization of messages in data pipelines.
- **Inter-Microservice Communication:** Enables language-agnostic data exchange between different microservices within an organization.
- **Long-Term Data Storage:** Its efficient format and schema evolution support make it suitable for archival and data warehousing needs.


Serialization
De-Serialization
Object
Stream of bytes
File
DB
Memory
Stream of bytes
Object