All About **ORC (Optimized Row Columnar)**

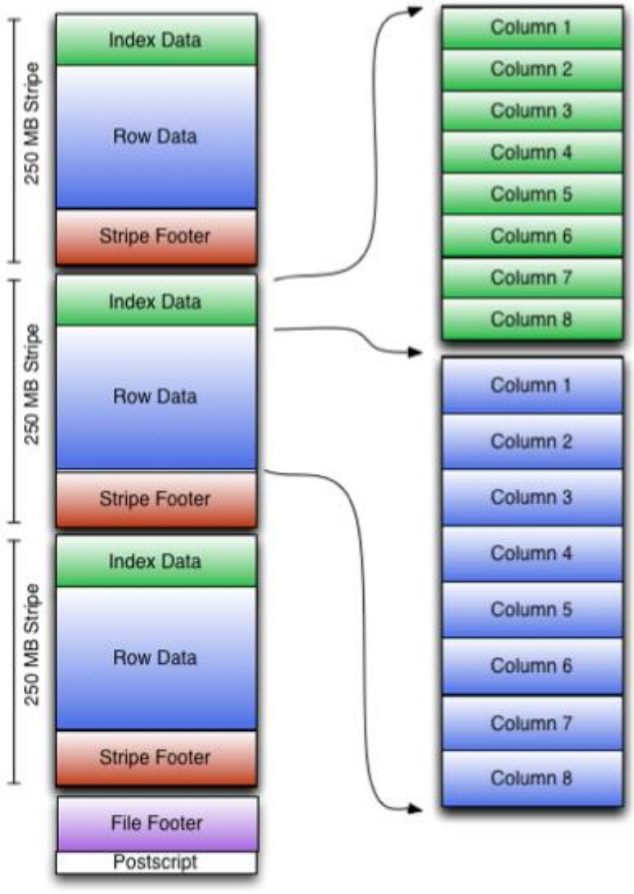
Snowflake for Big Data

Snowflake is an ideal platform for executing big data workloads using a variety of file formats, including **Parquet**, **Avro**, and **XML**. Snowflake makes it easy to ingest **semi-structured data** and combine it with structured and unstructured data. With Snowflake, you can specify compression schemes for each column of data with the option to add additional encoding at any time.

Using Snowflake, you can create and run modern integrated data applications, democratize data analytics so team members of all skill levels can make data-driven decisions, and develop new revenue streams based on data to help drive your business forward. Snowflake makes it possible to realize the full potential of your data, with the flexibility to choose which file format best meets your needs.

Snowflake provides users the ability to easily work with various forms of data including JSON, AVRO, XML, PARQUET, and ORC.

An ORC (Optimized Row Columnar) file is a data storage format designed for Hadoop and other big data processing systems. It is a columnar storage format, which means that the data is stored in a way that is optimized for column-based operations like filtering and aggregation.



All About **ORC (Optimized Row Columnar)**

ORC File Structure

Internally, ORC stores data in a series of stripes, where each stripe is a collection of rows. Each stripe is further divided into a series of data chunks, where each chunk stores the data for a specific set of columns. The chunks are compressed using a combination of techniques such as predicate filtering, dictionary encoding, and run-length encoding.

ORC also stores metadata about the file, such as the schema, at the end of the file. This metadata is used to quickly read the data without having to scan the entire file. Additionally, ORC can store indexes for specific columns, allowing for faster retrieval of specific rows.

One of the main advantages of using ORC files is that they offer significant performance improvements over row-based storage formats like text and avro. This is because, in a columnar storage format, the data for a single column is stored together, which makes it faster to read and process. Additionally, ORC files also support advanced features such as predicate pushdown, which allows the storage format to filter out unnecessary data before it is read into memory, further improving performance.

ORC files also offer good support for compression. It uses a number of compression algorithms such as Snappy, Zlib, Gzip etc. which reduces the storage space required to store the data. This is particularly useful when working with large datasets, as it can significantly reduce the cost of storing the data.

Another advantage of ORC files is that they support a wide range of data types, including complex types such as structs, maps, and arrays. This makes it easy to work with different kinds of data, and allows for more flexible data modeling.

To use ORC files with PySpark, you can simply read and write data using the spark.read.format(“orc”) and .write.format(“orc”) options, respectively.

ORC (Optimized Row Columnar) files are widely used in the data analytics industry, especially in big data and Hadoop ecosystems. Here's how ORC files are utilized:

1. Efficient Storage:

- **Columnar Storage**: ORC files store data in a columnar format, which allows for high compression and efficient data retrieval. This is beneficial when working with large datasets, as it reduces storage requirements and speeds up query performance.
- **Compression**: ORC files support multiple compression algorithms (like Zlib, Snappy, LZO), which further reduce the size of the data stored, making it more cost-effective.

2. Improved Performance:

- **Faster Query Execution**: Because ORC files store data in a columnar format, queries that only require a few columns can read just those columns, reducing the amount of data that needs to be read from disk.

- **Indexing**: ORC files automatically include lightweight indexes (like min, max, and sum) for each column, which allows for faster query processing by skipping unnecessary data blocks.

3. Integration with Big Data Tools:

- **Hive and Spark**: ORC is natively supported by Apache Hive and Apache Spark, two popular big data processing frameworks. This allows for seamless integration and efficient processing of ORC files within these ecosystems.
- **Data Lake Storage**: ORC files are often used in data lakes, where large volumes of raw data are stored and processed. Their efficient storage and retrieval capabilities make them ideal for such environments.

4. Scalability:

- **Handling Large Datasets**: ORC files are designed to handle petabytes of data, making them suitable for large-scale data analytics tasks. Their structure supports the parallel processing capabilities of modern distributed systems.

5. Data Analytics and BI Tools:

- **Support in ETL Pipelines**: ORC files are frequently used in ETL (Extract, Transform, Load) processes to store intermediate and final datasets. Their compatibility with BI (Business Intelligence) tools enables efficient data analysis and reporting.

In summary, ORC files are favored in the data analytics industry for their ability to efficiently store, compress, and quickly retrieve large datasets, making them essential in big data processing and analysis environments.

ORC (Optimized Row Columnar) files are typically generated from various data sources and processing tools within big data ecosystems. Here are the main sources from which ORC files are generated:

1. Apache Hive

- **SQL Queries**: ORC files are often generated as the output of SQL queries executed in Apache Hive. When data is inserted into Hive tables, users can specify the ORC file format to store the data.
- **Table Storage**: Hive tables can be created with the ORC format, so any data loaded into these tables will be stored as ORC files.

2. Apache Spark

- **DataFrame Writes**: In Apache Spark, DataFrames can be written to ORC format using Spark's DataFrame API. This is commonly done when transforming or processing large datasets that need to be stored efficiently.
- **Spark SQL**: Similar to Hive, Spark SQL allows users to save query results in ORC format.

3. Apache Pig

- **ETL Pipelines**: In Apache Pig, a data flow language used for ETL (Extract, Transform, Load) processes, users can store the results of their processing pipelines as ORC files.

4. Presto and Trino

- **Distributed SQL Engines**: Presto (and its successor, Trino) can generate ORC files as part of query execution when writing query results to storage. These engines often work with ORC files for their efficient storage and query performance in distributed environments.

5. Data Warehousing and ETL Tools

- **ETL Tools**: Various ETL tools and data integration platforms, such as Apache NiFi or Talend, can process and transform data into ORC format for efficient storage in data lakes or Hadoop Distributed File Systems (HDFS).
- **Data Warehouses**: Modern data warehouses like Amazon Redshift Spectrum or Google BigQuery can read and generate ORC files when integrating with Hadoop ecosystems.

6. Custom Scripts and Applications

- **Programmatic Generation**: ORC files can also be generated programmatically using APIs provided by Hadoop, Spark, or Hive. For example, developers can write applications in Java, Python, or Scala that use libraries like hive-exec to create ORC files directly.

7. Hadoop Distributed File System (HDFS)

- **MapReduce Jobs**: ORC files can be generated as output from Hadoop MapReduce jobs, where data processing tasks write their results in ORC format to HDFS.

8. Data Conversion Tools

- **Conversion Utilities**: Tools designed to convert data from other formats (e.g., CSV, JSON, Parquet) to ORC are also sources. These utilities are often used when migrating data to ORC for better performance and storage efficiency.

In essence, ORC files are typically generated by tools and platforms within the Hadoop ecosystem or big data environments where efficient data storage and fast retrieval are priorities.

What to choose — ORC or Parquet ?

- ORC is optimized for read-heavy workloads, as it includes features such as predicate pushdown, compression, and column pruning that can improve query performance.
- ORC also includes a feature called predicate pushdown, which allows Spark to skip over irrelevant data when querying large datasets.
  ORC supports a wide variety of compression codecs, including Zlib, Snappy, and LZO, which can help to reduce storage costs.

- Parquet is optimized for write-heavy workloads, as it includes features such as encoding and dictionary compression that can improve write performance.
- Parquet has better support for nested data structures, which can be useful for storing semi-structured data like JSON.
- Parquet has built-in support for predicate pushdown and column pruning, which can improve query performance.

When to choose which

- If your use case requires a lot of read-heavy workloads and you're working with a large dataset, then ORC is the better choice.
- If you're working with a write-heavy workload, need to store semi-structured data, or want to take advantage of encoding and dictionary compression, then Parquet is the better choice.
- It's worth noting that there are other file formats as well and the choice of format may depend on the specific requirements of your use case and the tools that you're using in your big data ecosystem.

Best practices for data storage and processing with ORC

1. Use predicate pushdown to optimize filter operations and reduce data read.
2. Use bucketing to group similar data together for faster query performance.
3. Use compression to reduce data storage costs and improve read performance.
4. Use vectorization to improve query performance by processing multiple rows at a time.
5. Use data partitioning to organize data by specific columns for efficient querying and management.
6. Use bloom filters to improve query performance by quickly identifying non-matching data.
7. Use data statistics to make better decisions about query optimization, such as choosing which columns to use for partitioning or bucketing.
8. Use data encoding to compress the data further and reduce storage costs.
9. Use caching to store frequently accessed data in memory for faster query performance.
10. Monitor and maintain the ORC data storage and process it regularly to ensure optimal performance.

Conclusion

In conclusion, ORC file format is an efficient storage format for big data processing system.It is columnar in nature, supports advanced features like predicate pushdown, data compression and supports wide range of data types.

References : https://www.snowflake.com/trending/avro-vs-parquet