



Hive File Formats

IMPORTANT

Copyright Infringement and Illegal Content Sharing Notice

All course content designs, video, audio, text, graphics, logos, images are Copyright© and are protected by India and international copyright laws. All rights reserved.

Permission to download the contents (wherever applicable) for the sole purpose of individual reading and preparing yourself to crack the interview only. Any other use of study materials – including reproduction, modification, distribution, republishing, transmission, display – without the prior written permission of Author is strictly prohibited.

Trendytech Insights legal team, along with thousands of our students, actively searches the Internet for copyright infringements. Violators subject to prosecution.



Data Storage Options

One of the most fundamental decisions to make when you are architecting a solution on Hadoop is determining how data will be stored in Hadoop.

Major considerations for Hadoop data storage include:

1. File Formats

2. Compression



Why do we need different file formats?

we want to do better in terms of:

**storage cost
processing speeds
IO costs**

As the data increases, the cost for storage and processing increases too.

The various Hadoop file formats have evolved as a way to ease these issues across a number of use cases.



Choosing an appropriate file format can have some significant benefits:

Faster read times

Faster write times

Splittable files

Schema evolution support

Advanced compression support

Most compatible platform

Some file formats are designed for general use. others are designed for more specific use cases. So there really is quite a lot of choice.



The file format in Hadoop roughly divided into two categories: row-oriented and column-oriented:

Row-oriented:

the whole row (all columns) of data from the disk has to be read in memory even if you need only a few columns. Row-oriented storage is suitable for situations where the entire row of data needs to be processed simultaneously.

Column-oriented:

The entire file is divided in several columns of data, and each column of data stored together

The column-oriented format makes it possible to skip unneeded columns when reading data, suitable for situations where only a small portion of the rows are needed.

ID	Name	Department
1	emp1	d1
2	emp2	d2
3	emp3	d3

For this table, the data in a row-wise storage format will be stored as follows:

1	emp1	d1	2	emp2	d2	3	emp3	d3
---	------	----	---	------	----	---	------	----

Whereas, the same data in a Column-oriented storage format will look like this:

1	2	3	emp1	emp2	emp3	d1	d2	d3
---	---	---	------	------	------	----	----	----



Text file format

We can store data using standard file formats in Hadoop for example, text files (such as comma-separated value [CSV] or XML), however it's preferable to use one of the Hadoop-specific container formats which we will talk about shortly.

It is a row based file format.

example csv file.

Not ideal in Hadoop as it can take a lot of storage and very slow in processing.

you'll want to select a compression format for the files, since text files can very quickly consume considerable space on your Hadoop cluster.



Also, keep in mind that there is an overhead of type conversion associated with storing data in text format.

For example, storing 1234 in a text file and using it as an integer requires a string-to-integer conversion during reading, and vice versa during writing.

It also takes up more space to store 1234 as text than as an integer.



XML & Json - Structured text data

A more specialized form of text files is structured formats such as XML and JSON.

These types of formats can present special challenges with Hadoop since splitting XML and JSON files for processing is tricky, and Hadoop does not provide a built-in InputFormat for either.



Specialized File Formats

There are several Hadoop-specific file formats that were specifically created to work well with Big Data Technologies.

These Hadoop-specific file formats include serialization formats like Avro and columnar formats such as Orc and Parquet.

These file formats have differing strengths and weaknesses, but all share the following characteristics that are important for Hadoop applications:

Splittable

The ability to split a file for processing by multiple tasks is of course a fundamental part of parallel processing, and is also key to leveraging Hadoop's data locality feature.

Agnostic compression

The file can be compressed with any compression codec, without readers having to know the codec. This is possible because the codec is stored in the header metadata of the file format.



Avro is a row-based storage format for Hadoop which is widely used as a serialization platform.

Avro stores the schema in JSON format making it easy to read and interpret by any program.

The data itself is stored in a binary format making it compact and efficient.

Avro is a language-neutral data serialization system. It can be processed by many languages (currently C, C++, C#, Java, Python, and Ruby).

A key feature of Avro is the robust support for data schemas that changes over time, i.e. schema evolution. Avro handles schema changes like missing fields, added fields and changed fields. A critical feature if your data has the potential to change.



This format is the ideal candidate for storing data in a data lake landing zone, because:

- 1. Data from the landing zone is usually read as a whole for further processing by downstream systems (the row-based format is more efficient in this case).**
- 2. Downstream systems can easily retrieve table schemas from files (there is no need to store the schemas separately in an external meta store).**
- 3. Any source schema change is easily handled (schema evolution).**

Avro is typically the format of choice for write-heavy workloads given its easy to append new rows.



Avro is language-neutral data serialization

Avro stores the schema in the header of the file so data is self-describing

Avro formatted files are splittable and compressible and hence it's a good candidate for data storage in Hadoop ecosystem.



The Optimized Row Columnar (ORC) file format is a column based file formats and provides a highly efficient way to store data in a compact manner.

The ORC format provides the following features and benefits:

Provides lightweight, always-on compression provided by type-specific readers and writers such as dictionary encoding, bit packing, delta encoding, and run length encoding – resulting in dramatically smaller files. ORC also supports the use of zlib, LZO, or Snappy to provide further compression.

Allows predicates to be pushed down to the storage layer so that only required data is brought back in queries.



Hive type support including DateTime, decimal, and the complex types (struct, list, map, and union)

Is a splittable storage format.

Metadata stored using Protocol Buffers, which allows the addition and removal of fields.

The file is broken into three parts- Header, Body, and Tail.

1. The Header consists of the bytes “ORC” to support tools that want to scan the front of the file to determine the type of the file.

2. The body of the file is divided into stripes. Each stripe is self contained and may be read using only its own bytes combined with the file’s Footer and Postscript.



The default stripe size is 250 MB. Large stripe sizes enable large, efficient reads from HDFS.

Stripes have three sections:

a set of indexes for the rows within the stripe.

The data itself

stripe footer - contains the encoding of each column

3. Tail consists of

File Footer

The file footer contains a list of stripes in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum.



Postscript

At the end of the file a postscript holds compression parameters and provides a way to interpret rest of the file.

Indexes

ORC provides three level of indexes within each file:

file level - statistics about the values in each column across the entire file

stripe level - statistics about the values in each column for each stripe

row level - statistics about the values in each column for each set of 10,000 rows within a stripe.

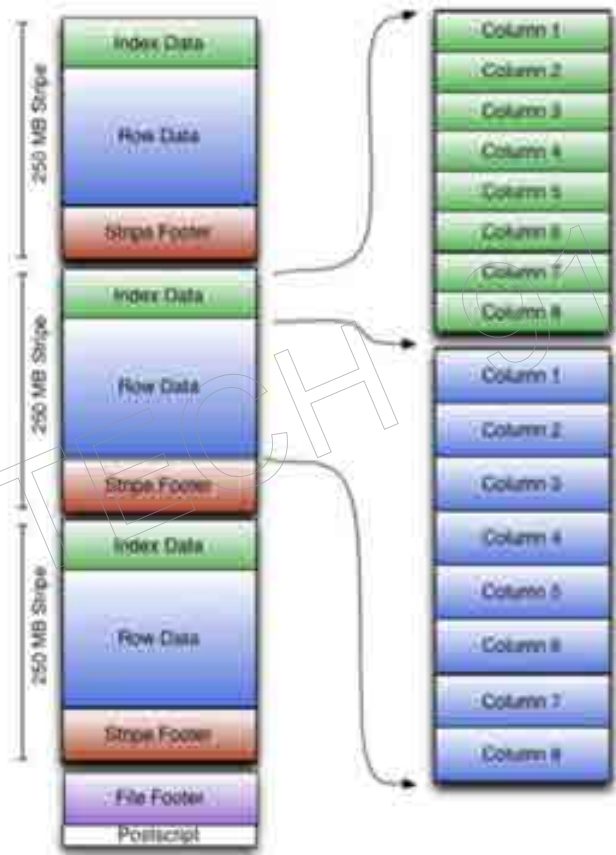


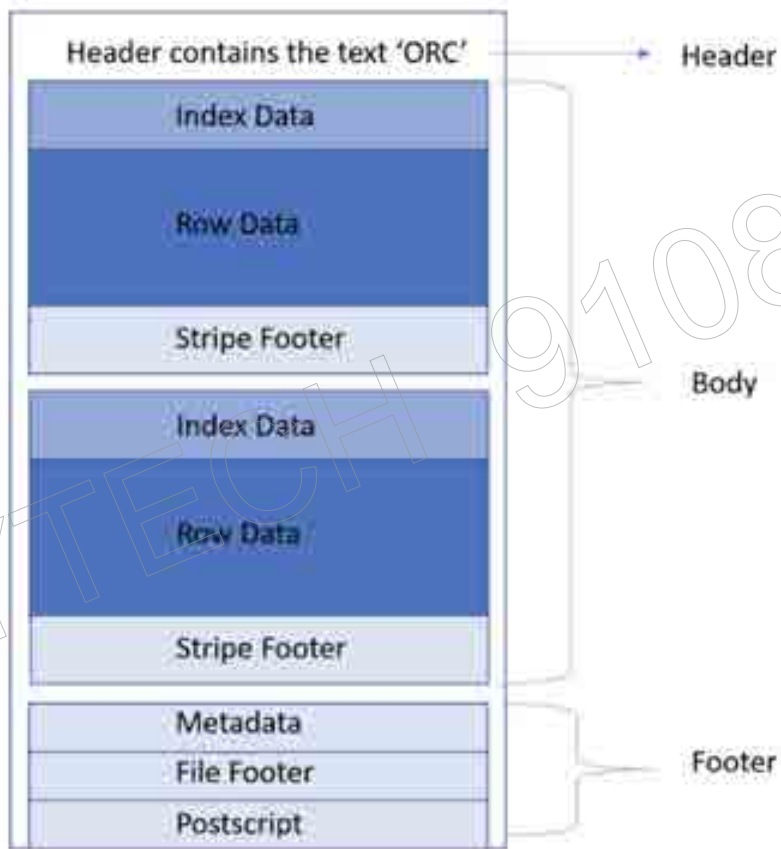
The file and stripe level column statistics are in the file footer so that they are easy to access to determine if the rest of the file needs to be read at all.

Row level indexes include both the column statistics for each row group and the position for seeking to the start of the row group.

The process of reading an ORC file works backwards through the file. The ORC reader reads the last 16k bytes of the file with the hope that it will contain both the Footer and Postscript sections.

A drawback of ORC was that it was designed specifically for Hive, and was not a general-purpose storage format that can be used with non-Hive MapReduce interfaces such as Java. However now lot of these shortcomings are covered now.







A generic column-oriented storage format based on Google's Dremel. Especially good at handling deeply nested data.

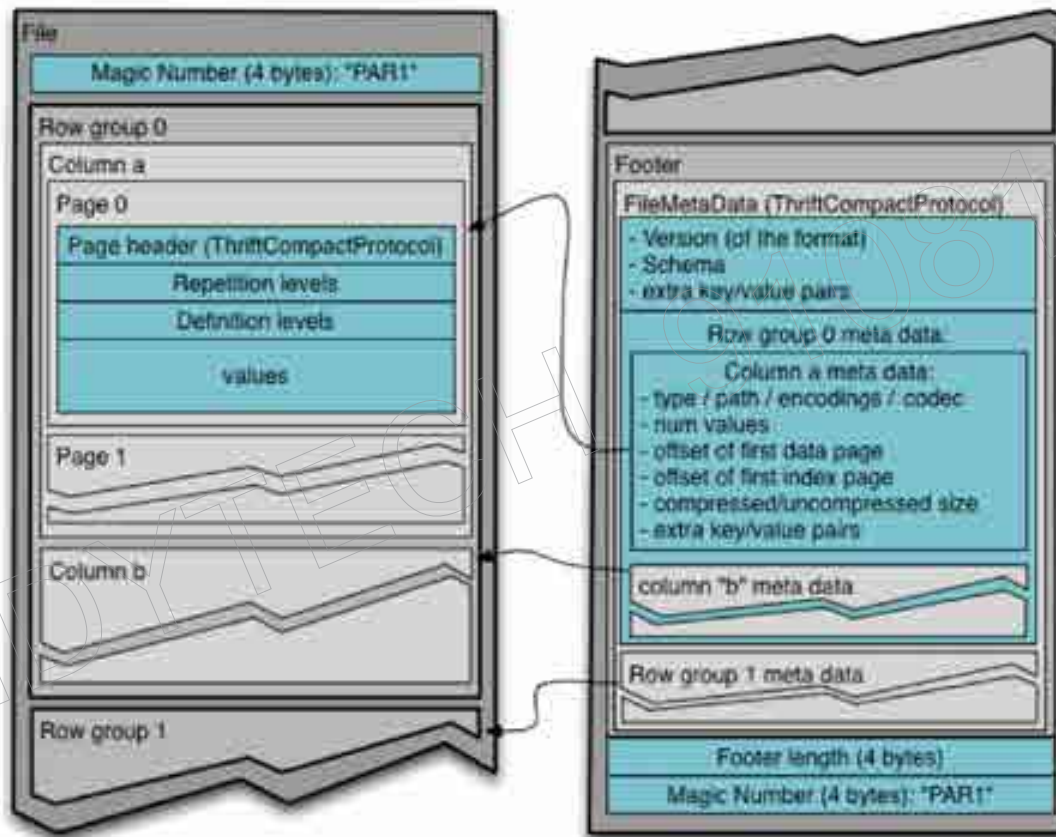
Parquet shares many of the same design goals as ORC, but is intended to be a general-purpose storage format for Hadoop.

Similar to ORC files, Parquet allows for returning only required data fields, thereby reducing I/O and increasing performance.

Provides efficient compression.

Stores full metadata at the end of files, so Parquet files are self-documenting.

Uses efficient and extensible encoding schemas—for example, bit-packaging/run length encoding (RLE).



Header

Row group

Row group

.....

Row group

Footer

<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>

<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>

<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>



It is more efficient when you need to query a few columns from a table. It will read only the required columns since they are adjacent thus minimizing IO.

It handles nested data structures in a flat columnar format. This means that in a Parquet file format, even the nested fields can be read individually without the need to read all the fields in the nested structure.

To understand the Parquet file format in Hadoop you should be aware of the following terms-

Row group: A logical horizontal partitioning of the data into rows. A row group consists of a column chunk for each column in the dataset.



Column chunk: A chunk of the data for a particular column. These column chunks live in a particular row group and are guaranteed to be contiguous in the file.

Page: Column chunks are divided up into pages written back to back. The pages share a common header and readers can skip the page they are not interested in.

Here, the Header just contains a magic number “PAR1” (4-byte) that identifies the file as Parquet format file.

Footer contains the following-

File metadata- The file metadata contains the locations of all the column metadata start locations.

length of file metadata (4-byte)

magic number “PAR1” (4-byte)



COMPARISONS BETWEEN DIFFERENT FILE FORMATS

AVRO vs PARQUET

AVRO is a row-based storage format whereas PARQUET is a columnar based storage format.

PARQUET is much better for analytical querying i.e. reads and querying are much more efficient than writing.

Write operations in AVRO are better than in PARQUET.



AVRO is much mature than PARQUET when it comes to schema evolution.

PARQUET only supports schema append whereas AVRO supports a much-featured schema evolution i.e. adding or modifying columns.

PARQUET is ideal for querying a subset of columns in a multi-column table.

AVRO is ideal in case of ETL operations where we need to query all the columns.



ORC vs PARQUET










PARQUET is more capable of storing nested data.

ORC is more capable of Predicate Pushdown.

ORC supports ACID properties.

ORC is more compression efficient.

BIG DATA FORMATS COMPARISON

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read



We have learnt a few more hive Optimizations

Happy Learning!!!



5 Star Google Rated
Big Data Course

LEARN FROM THE EXPERT



9108179578

Call for more details



Follow US

Trainer Mr. Sumit Mittal

Phone 9108179578

Email trendytech.sumit@gmail.com

Website <https://trendytech.in/courses/big-data-online-training/>

LinkedIn <https://www.linkedin.com/in/bigdatabysumit/>

Twitter @BigdataBySumit

Instagram bigdatabysumit

Facebook <https://www.facebook.com/trendytech.in/>

Youtube https://www.youtube.com/channel/UCbTggJVf0NDTfWX-C_gUGSg