

# FILE FORMATS

- AVRO
- PARQUET
- ORC

H. Brunecky,

Uno dei più grandi colli di battaglia in ambito Big Data è quello di leggere, individuazione e spostare, dati.

Per queste ragioni è necessario scegliere il formato di dati adatto alla situazione per ottenerne benefici quali:

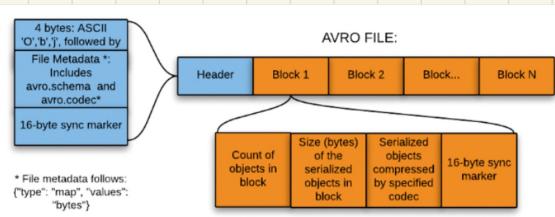
- 1. Lettura più veloci
- 2. Scrittura più veloci
- 3. file splitabili
- 4. Soggetto allo schema evolution
- 5. Supporto alla compressione

## AVRO FILE

E' un formato **row-based** ampiamente utilizzato. Lo schema è salvato in **JSON** rendendo la lettura e l'interpretazione facile.

I dati invece vengono salvati in modo **binario** rendendolo compatto ed efficiente, e una serializzazione neutrale e usabile in molti linguaggi.

Unico punto di forza è lo **schema evolution**.



Avro è l'ideale per salvare dati in una **landing zone** poiché:

1. Si salva nelle landing zone (staging); i dati vengono salvati così come sono, per essere elaborati per intero insieme.
2. I sistemi che si occupano dell'elaborazione possono recuperare gli schemi facilmente.
3. I cambi di schema sono gestiti facilmente.

## PARQUET FILE

Parquet salvo: dati in **formato colonna**, ideali per i **tipi strutturati**. E' molto più efficiente in termini di storage e performance.

Ideale quando le query/interrogazioni sono **specifiche su colonne**.

## COLUMNAR STORAGE FORMAT

In questo tipo il valore di ogni colonna dello stesso tipo vengono ordinati insieme.

For example, if there is a record comprising ID, employee Name, and Department, then all the values for the ID column will be stored together, values for the Name column together, and so on. If we take the same record schema as mentioned above, having three fields ID (int), NAME (varchar), and Department (varchar), the table will look something like this:

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
---	------	----	---	------	----	---	------	----

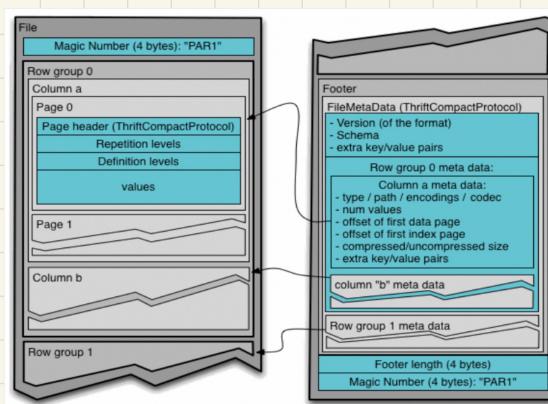
Whereas, the same data in a Column-oriented storage format will look like this:

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

Questo permette di velocizzare le operazioni di I/O, poiché se c'è necessità di querire solo una colonna verranno letti solo i dati di quella colonna, tutti adiacenti fra loro.

Una delle fortezze principali di PARQUET è quella di poter salvare dati strutturati in modo colonna, rendendo possibile

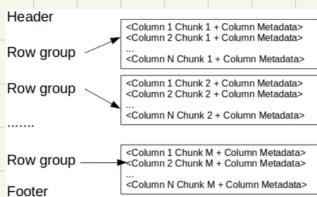
La lettura dei singoli campi è indipendente.



**ROW GROUP:** posizione logica orizzontale in righe. Una row group è composta da column chunk, uno per ogni colonna.

**COLUMN CHUNK:** un blocco di dati per una determinata colonna. Sono continue nel file.

**PAGE:** i column chunk sono divisi in page scritte una dopo l'altra. Esse condividono un header, in modo tale da poter filtrare.



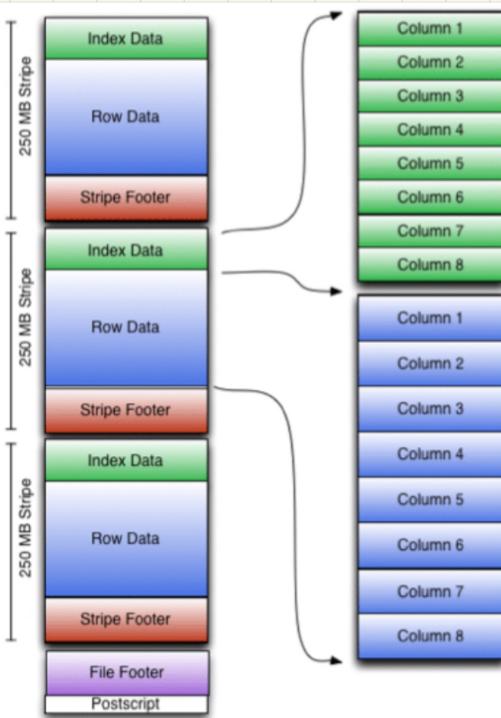
Il footer contiene metadati

- lunghezza metadati (4 byte)
- magic number "PAR1" (4 byte)

## ORC

E' ideale per memorizzare dati compressi permettendo di saltare parti a cui non si è interessati. I vantaggi quindi:

- Un singolo file di output per ogni task.
- Supporto completo per tipi mixti
- Lettura concorrente dello stesso file.
- Capacità di spartire file senza markers.
- Stivare un upper bound e un heap memory basato sul file footer
- I metadati sono salvi usando Protocols Buffers che permettono di aggiungere e rimuovere campi.



ORC è una collezione di righe in UN file in cui sono scritte in modo colomnare.

Ogni file contiene un gruppo di righe (detti STRIPES) e un FOOTER. In più alla fine del file c'è un POSTSCRIPT che contiene i parametri di compressione.

Il FOOTER contiene:

1. una lista di STRIPES
2. Numero di righe per STRIPE
3. Tipo di colonna

Lo STRIPE FOOTER contiene le stream locations. Contiene inoltre aggregazioni (sum, avg, count, ...)

## COMPARAZIONE

### AVRO vs. PARQUET

1. AVRO is a row-based storage format, whereas PARQUET is a columnar-based storage format.
2. PARQUET is much better for analytical querying, i.e., reads and querying are much more efficient than writing.
3. Writing operations in AVRO are better than in PARQUET.
4. AVRO is much matured 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.
5. PARQUET is ideal for querying a subset of columns in a multi-column table. AVRO is ideal in the case of ETL operations, where we need to query all the columns.

### ORC vs. PARQUET

1. PARQUET is more capable of storing nested data.
2. ORC is more capable of Predicate Pushdown.
3. ORC supports ACID properties.
4. ORC is more compression efficient.