**Sqoop:**

**Compression**

Sometimes we would need to keep imported data on HDFS in compressed state in order to reduce the overall disk utilization. As sqoop is based on Map Reduce execution, it inherits Hadoop's all compression features. It allows is to save imported data into various compressed formats e.g. GZIP or BZ2 etc. We can execute following query to use by default gzip compression,

Reading and processing compressed file has no issue; we can just read it as normal file;

**But there may be bit of performance degrade with compressed file.(which is very minimum)**

**Say I used BZip2Codec compression algorithm**

1. Default Order\_items data size without compression

5408880 - without compression

1. **Best compression what I got is with BZip2Codec compression algorithm**

811769 - BZip2Codec (file saved as - **part-m-00000.bz2**)

sqoop import \

--connect "jdbc:mysql://nn01.itversity.com/retail\_db" \

--username=retail\_dba \

--password=itversity \

--table order\_items \

--target-dir /user/madhanrajuj2/sqoop\_import/compression/BZip2Codec \

-m 1 \

--compress \

**--compression-codec org.apache.hadoop.io.compress.BZip2Codec \**

--outdir java\_files

1. If we use –compress parameter without using –compression-codec (compression algorithem); **by default it uses gzip**

1030482 -gzip (default compression) -- (/**part-m-00000.gz** - saved as .gz)

sqoop import \

--connect "jdbc:mysql://nn01.itversity.com/retail\_db" \

--username=retail\_dba \

--password=itversity \

--table order\_items \

--target-dir /user/madhanrajuj2/sqoop\_import/order\_items/compress\_codec \

-m 1 \

--compress \

--outdir java\_files

1. 1825926 – **SnappyCodec** (file saved as .snappy)

sqoop import \

--connect "jdbc:mysql://nn01.itversity.com/retail\_db" \

--username=retail\_dba \

--password=itversity \

--table order\_items \

--target-dir /user/madhanrajuj2/sqoop\_import/order\_items/compress\_codec \

-m 1 \

--compress \

--**compression-codec org.apache.hadoop.io.compress.SnappyCodec**

--outdir java\_files

**hadoop file formats**

**-avrodatafile**

- Platform independent

-Compact, fast, binary data formats

-when Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program

if the program reading the data expects a different schema this can be easily resolved, since both schema are present.

- we can use same avro file with old and new applications;

In old we have 5 data fields

In new we have 6 data fields; same file can be used with new app; by adding default parameter in avro schema(.avsc)

**avro-tools**

**------------------**

**Video 82:**

<https://www.youtube.com/watch?v=loluXI1KyFc&list=PLf0swTFhTI8rJvGpOp-LujOcpk-Rlz-yE&index=82>

* We can use “**avro-tools** “ to extract schema

Command: **avro-tools getschema part-m-00000.avro >**

----converts avro data file to in to json file

Command: **avro-tools tojson part-m-00000.avro** ----converts avro in to

Command: **avro-tools getschema part-m-00000.avro > departments.avsc** --- to extract schema of department

cat departments.avsc

**avro-tools fromjson departments.json --schema-file departments.avsc > departments.avro** ---to convert JSon file to avro data file (use “> departments.avro” at the end)

**-if we modify meta data and create a new table on existing avro data and insert data in to avro table; it will create separate file without touching existing avor data file; so that any other table refereeing to same data can still use the avro data without any issue**

**Video 83:**

<https://www.youtube.com/watch?v=2-27UvXCOyo&index=83&list=PLf0swTFhTI8rJvGpOp-LujOcpk-Rlz-yE>

**Video 84:**

<https://www.youtube.com/watch?v=DXmmTNOp9bk&list=PLf0swTFhTI8rJvGpOp-LujOcpk-Rlz-yE&index=84>

**Video 85: - evolving a avro schema**

<https://www.youtube.com/watch?v=mifQGz-6UY0&list=PLf0swTFhTI8rJvGpOp-LujOcpk-Rlz-yE&index=85>

**-if we modify meta data and create a new table on existing avro data and insert data in to avro table; it will create separate file without touching existing avor data file; so that any other table refereeing to same data can still use the avro data without any issue**

Note: **impala we cannot insert data in to table using avro format**

**to read AVRO for interview\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**----------------------**

<https://www.youtube.com/watch?v=WHncMILyVYs&index=4&list=PLnQL3JwXMDv3ToYko3d5AJ0V6a_RPS-ml>

**schema and data is stored to gether(Schema+data)**

**Schema is in json format**

**Data is in binary format(it also reduces the size of the file as it is stored in binary format)**

**Avro files are language Neutral**

**--> data may be created by Java but same data can be read by C++ language**

**--> writing is Serialization**

**--> read is a de-Serialization**

**Schema Evaluation --**

**--adding a fields**

**--removing a fields**

**avro clearly handles schema changes like**

**1. missing fields**

**2. Newely added fields**

**3. Changed fields**

**Because of this**

**1. old program can read data[based on old schema]**

**2. New program can read data[based on new schema]**

**Parquet file format:**columnar format data storage

* columnar file format
* we can achieve better compression enabling faster queries.
* generally slower to write than non-columnar file formats.

**(when we are importing via sqoop command)**

One note on Parquet file support with Hive... It is very important that Parquet column names are lowercase. If your Parquet file contains mixed case column names, Hive will not be able to read the column and will return queries on the column with null values *and not log any errors*. Unlike Hive, Impala handles mixed case column names. A truly perplexing problem when you encounter it!

**Advantages:**

**-data can be highly compressed**

**-** Another benefit is that because a column-based DBMSs is self-indexing, it uses less disk space

**Advantages of using Parquet**

There are several advantages to columnar formats.

* Organizing by column allows for better compression, as data is more homogeneous. The space savings are very noticeable at the scale of a Hadoop cluster.
* I/O will be reduced as we can efficiently scan only a subset of the columns while reading the data. Better compression also reduces the bandwidth required to read the input.
* As we store data of the same type in each column, we can use encoding better suited to the modern processors’ pipeline by making instruction branching more predictable.

in a row-oriented database management system, the data would be stored like this:  1,Doe,John,8000;2,Smith,Jane,4000;3,Beck,Sam,1000;

In a column-oriented database management system, the data would be stored like this:  1,2,3;Doe,Smith,Beck;John,Jane,Sam;8000,4000,1000;

One of the main benefits of a columnar database is that data can be highly [compressed](http://searchstorage.techtarget.com/definition/compression). The compression permits columnar operations — like MIN, MAX, SUM, COUNT and AVG— to be performed very rapidly.  Another benefit is that because a column-based DBMSs is self-indexing, it uses less disk space than a relational database management system ([RDBMS](http://searchsqlserver.techtarget.com/definition/relational-database-management-system)) containing the same data.

create table ptest (a INT, b INT)

stored as parquet;

We cannot load  text file directly into parquet table , we should first create an alternate table to store the text file and use insert overwrite command to write the data in parquet format.

hive> create table te3 (x int, y int)

> row format delimited

> FIELDS TERMINATED BY ','

> STORED AS TEXTFILE;

hive> LOAD DATA LOCAL INPATH '/home/cloudera/test/' OVERWRITE INTO TABLE te3;

insert overwrite table ptest select \* from te3;

**Important Note:**

**----------------------------------**

**For compression we can use SnappyCodec algorithem with Parquet file**

**BZip2Codec – can not be used with Parquet file format**

**With avro we can use BZip2Codec algorithem for better compression**

**sqoop import \**

**-m 1 \**

**--connect "jdbc:mysql://nn01.itversity.com/retail\_db" \**

**--username retail\_dba \**

**--password itversity \**

**--target-dir /user/madhanrajuj2/sqoop\_import/testingSize/orders\_avro7 \**

**--table orders \**

**--as-avrodatafile \**

**--compress \**

**--compression-codec org.apache.hadoop.io.compress.BZip2Codec \**

**--outdir java\_file**

**sqoop import \**

**-m 1 \**

**--connect "jdbc:mysql://nn01.itversity.com/retail\_db" \**

**--username retail\_dba \**

**--password itversity \**

**--target-dir /user/madhanrajuj2/sqoop\_import/testingSize/orders\_parquet6 \**

**--table orders \**

**--as-parquetfile \**

**--compress \**

**--compression-codec org.apache.hadoop.io.compress.SnappyCodec \**

**--outdir java\_file**

**avro --**

**1779793 - withoutcompression**

**653049 - with SnappyCodec**

**322686 - with BZip2Codec**

**322686 - with BZip2Codec**

**Parquet --**

**488059 - with SnappyCodec**

### **ORC Files**

**O**ptimized **R**ow **C**olumnar(ORC)

* columnar file format
* we can achieve better compression enabling faster queries.
* generally slower to write than non-columnar file formats.
* pro: high level of compression
* pro: high level of speed on read/write
* con: schema is partially-embedded in data

**http://inquidia.com/news-and-info/hadoop-file-formats-its-not-just-csv-anymore**

### **How to choose a file format?**

As discussed, each file format is optimized by purpose. Your choice of format is driven by your use case and environment. Here are the key factors to consider:

* **Hadoop Distribution**- Cloudera and Hortonworks support/favor different formats
* **Schema Evolution**- Will the structure of your data evolve?  In what way?
* **Processing Requirements** - Will you be crunching the data and with what tools?
* **Read/Query Requirements**- Will you be using SQL on Hadoop?  Which engine?
* **Extract Requirements**- Will you be extracting the data from Hadoop for import into an external database engine or other platform?
* **Storage Requirements**- Is data volume a significant factor?  Will you get significantly more bang for your storage buck through compression?

So, with all the options and considerations are there any obvious choices?  If you are storing intermediate data between MapReduce jobs, then Sequence files are preferred. If query performance against the data is most important, ORC (HortonWorks/Hive) or Parquet (Cloudera/Impala) are optimal --- but these files will take longer to write. (We’ve also seen order of magnitude query performance improvements when using Parquet with Spark SQL.) Avro is great if your schema is going to change over time, but query performance will be slower than ORC or Parquet. CSV files are excellent if you are going to extract data from Hadoop to bulk load into a database.

Because you will likely have multiple use cases for your Hadoop data, you will use multiple file formats. It is quite common to store the same (or very similar) data in multiple file formats to support varied processing, query and extract requirements. Embrace diversity!