YCBS-257 - Data at Scale

Workshop 6

Part 2-a

Comparing File Formats: Data Preparation

General Instructions:

The purpose of this workshop is to familiarize you with the most popular Data at Scale file formats, **Avro**, **Parquet, and ORC.** It aims to help you to understand each format pros and cons to choose the best one for your use case and optimize storage space and processing time.

Online resources:

https://avro.apache.org/
https://parquet.apache.org/
https://orc.apache.org/

Note: *This workshop is divided into 5 parts and should be accomplished in the same order.*

Exercise 1-a: Data Preparation.

In this exercise you will prepare the initial data to be used in the next parts of this workshop. This will be our base line to convert and compare different file formats. You will use the pagecounts-20160801-000000 dataset (which is a partial page views log from Wikipedia).

This dataset has four columns:

- Project code
- Page name
- Page views (number of views)
- Bytes (length of the page in bytes)

1. Create a new Zeppelin note and select the Hive interpreter (`%hive`).
2. Place `pagecounts-20160801-000000` on HDFS.

```
hdfs dfs -mkdir -p /workshops/fformats/lab01/data

hdfs dfs -put /home/training/Data/pagecounts-20160801-000000
/workshops/fformats/lab01/data
```

3. Show the size of the file on HDFS. (*This is important as we need to compare formats size later*)

```
hdfs dfs -du -h /workshops/fformats/lab01/data
```

```
324.5 M  324.5 M  /workshops/fformats/lab01/data/pagecounts-20160801-000000
```

4. Create a (new) database.

```
create database if not exists fformats;
```

5. Drop the staging table `pagecounts` (if exists, this table *will be used later)*

```
drop TABLE if exists fformats.pagecounts;      (should not exists right now)
```

6. Create a new User Managed table named `pagecounts` to read the input file.

```
CREATE EXTERNAL TABLE fformats.pagecounts (projectcode STRING, pagename STRING, pageviews STRING, bytes STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' LINES TERMINATED BY '\n'STORED AS TEXTFILE LOCATION '/workshops/fformats/lab01/data';
```

7. Show the first 10 rows from the table.

```
select * from fformats.pagecounts limit 10;
```

8. Compute statistics for the `pagecounts` table.

```
Analyze table fformats.pagecounts compute statistics;
```

9. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts;
```

10. Report the size of the pagecounts table.

```
SHOW TBLPROPERTIES fformats.pagecounts ('totalSize');
```

340304295

11. Report the row count from the pagecounts table.

```
SHOW TBLPROPERTIES fformats.pagecounts ('numRows');
```

6270942

Workshop 6

Part 2-b

File Formats: AVRO Format

General Instructions:

Avro is a row-oriented remote procedure call and data serialization framework developed within Apache's Hadoop project. It uses JSON for defining data types and protocols and serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide both a serialization format for persistent data, and a wire format for communication between Hadoop nodes, and from client programs to the Hadoop services. Avro uses a schema to structure the data that is being encoded.

Online resources:

https://avro.apache.org/

Note: *This workshop is divided into 5 parts and should be accomplished in the same order.*

Exercise 1-b: Converting Data to AVRO.

In this part of the exercise, you will convert the initial dataset you prepared in Part 1-a to Avro format.

In this activity you will:

- Create an:
  - Avro table with no compression.
  - Avro table and use `Snappy` compression codec.
  - Avro table and use `Defalte` compression codec.
- Compare the properties of these tables to the initial dataset.
- Explore output metadata and extract Avro schema.

AVRO Table with No Compression

1. Create a new Zeppelin note and choose the Hive interpreter (`%hive`).
2. By default, Output Compression is disabled in Hive, setting this property to `true` will enable compression using `Deflate` Codec.

```
set hive.exec.compress.output=false;
```

3. Create a new User Managed table named `pagecounts_avro` and populated from the initial table.

```
create table fformats.pagecounts_avro stored as avro LOCATION
'/workshops/fformats/lab01/avro' as select * from fformats.pagecounts;
```

4. Compute statistics for the `pagecounts_avro` table.

```
Analyze table fformats.pagecounts_avro compute statistics;
```

5. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_avro;
```

6. Show the first 10 rows from the table.

```
select * from fformats.pagecounts_avro limit 10;
```

7. Report the size of the `pagecounts_avro` table.

```
SHOW TBLPROPERTIES fformats.pagecounts_avro ('totalSize');
```

366391926

AVRO Table with Snappy Compression

8. Enable Output Compression and set the compression codec to Avro.

```
set hive.exec.compress.output=true;
set avro.output.codec=snappy;
```

9. Create a new User Managed table named `pagecounts_avro_snappy` and populated from the initial table.

```
create table fformats.pagecounts_avro_snappy stored as avro LOCATION
'/workshops/fformats/lab01/avro_snappy' as select * from
fformats.pagecounts;
```

10. Compute statistics for the `pagecounts_avro` table.

```
Analyze table fformats.pagecounts_avro_snappy compute statistics;
```

11. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_avro_snappy;
```

12. Report the size of the `pagecounts_avro_snappy` table.

```
SHOW TBLPROPERTIES fformats.pagecounts_avro_snappy ('totalSize');
```

147242278

AVRO Table with Deflate Compression

13. Enable Output Compression and set the compression codec to `Deflate`.

```
set hive.exec.compress.output=true;
```

14. Create a new User Managed table named `pagecounts_avro_Deflate` and populated from the initial table.

```
create table fformats.pagecounts_avro_Deflate stored as avro LOCATION
'/workshops/fformats/lab01/avro_Deflate' as select * from
fformats.pagecounts;
```

15. Compute statistics for the `pagecounts_avro_Deflate` table.

```
Analyze table fformats.pagecounts_avro_Deflate compute statistics;
```

16. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_avro_Deflate;
```

17. Report the size of the `pagecounts_avro_Deflate` table.

```
SHOW TBLPROPERTIES fformats.pagecounts_avro_Deflate ('totalSize');
```

94318706

Explore AVRO Files Metadata

18. List output directories on HDFS.

```
hdfs dfs -ls -R /workshops/fformats/lab01/avro

hdfs dfs -ls -R /workshops/fformats/lab01/avro_snappy

hdfs dfs -ls -R /workshops/fformats/lab01/avro_deflate
```

19. Get the size on HDFS for each output directory.

```
hdfs dfs -du -h -s /workshops/fformats/lab01/avro

hdfs dfs -du -h -s /workshops/fformats/lab01/avro_snappy

hdfs dfs -du -h -s /workshops/fformats/lab01/avro_deflate
```

20. Show metadata of an Avro bucket.

```
hadoop jar avro-tools getmeta
/workshops/fformats/lab01/avro_snappy/000000_0

hadoop jar avro-tools getmeta
/workshops/fformats/lab01/avro_deflate/000000_0
```

21. Extract the Avro file schema.

```
hadoop jar avro-tools getschema
/workshops/fformats/lab01/avro_snappy/000000_0
```

```
{
  "type" : "record",
  "name" : "pagecounts_avro_snappy",
  "namespace" : "fformats",
  "fields" : [ {
    "name" : "projectcode",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "pagename",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "pageviews",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "bytes",
    "type" : [ "null", "string" ],
    "default" : null
  } ]
}
```

Workshop 6

Part 2-c

File Formats: Parquet Format

General Instructions:

Apache Parquet is a column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. Apache Parquet is designed to be a common interchange format for both batch and interactive workloads. It is like other columnar-storage file formats available in Hadoop, namely RCFile and ORC.

Online resources:

https://parquet.apache.org/

Note: *This workshop is divided into 5 parts and should be accomplished in the same order.*

Exercise 2-c: Converting Data to Parquet.

In this part of the exercise, you will repeat the steps of the previous part to convert the initial dataset to Parquet format.

In this activity you will:

- Create a Parquet table with no compression.
- Create two Parquet tables and use two compression codecs: `Snappy` and `GZip`
- Compare the properties of these tables to the initial dataset.
- Explore output metadata and extract Parquet schema.

PARQUET Table with No Compression

1. Create a new Zeppelin note and choose the Hive interpreter (`%hive`).
2. By default, Output Compression is disabled in Hive, keep this setting.

```
set hive.exec.compress.output=false;
```

3. Create a new User Managed table named `pagecounts_parquet` and populated from the initial table.

```
create table fformats.pagecounts_parquet stored as parquet LOCATION
'/workshops/fformats/lab01/parquet' as select * from fformats.pagecounts;
```

4. Compute statistics for the pagecounts_parquet table.

```
Analyze table fformats.pagecounts_parquet compute statistics;
```

5. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_parquet;
```

6. Show the first 10 rows from the table.

```
select * from fformats.pagecounts_parquet limit 10;
```

7. Report the size of the pagecounts_parquet table.

```
SHOW TBLPROPERTIES fformats.pagecounts_parquet ('totalSize');
```

325611113

PARQUET Table with Snappy Compression

8. Enable Output Compression and set the compression codec to Snappy.

```
set hive.exec.compress.output=true;
```

9. Create a new User Managed table named pagecounts_parquet_snappy and populated from the initial table.

```
create table fformats.pagecounts_parquet_snappy stored as parquet LOCATION
'/workshops/fformats/lab01/parquet_snappy' TBLPROPERTIES
('parquet.compression'='SNAPPY') as select * from fformats.pagecounts;
```

10. Compute statistics for the pagecounts_parquet_snappy table.

```
Analyze table fformats.pagecounts_parquet_snappy compute statistics;
```

11. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_parquet_snappy;
```

12. Report the size of the pagecounts_parquet_snappy table.

```
SHOW TBLPROPERTIES fformats.pagecounts_parquet_snappy ('totalSize');
```

129128553

PARQUET Table with GZip Compression

1. Enable Output Compression and set the compression codec to GZip.

```
set hive.exec.compress.output=true;
```

2. Create a new User Managed table named pagecounts_parquet_gzip and populated from the initial table.

```
create table fformats.pagecounts_parquet_gzip stored as parquet LOCATION
'/workshops/fformats/lab01/parquet_gzip' TBLPROPERTIES
('parquet.compression'='GZip') as select * from fformats.pagecounts;
```

3. Compute statistics for the pagecounts_parquet_gzip table.

```
Analyze table fformats.pagecounts_parquet_gzip compute statistics;
```

4. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_parquet_gzip;
```

5. Report the size of the pagecounts_parquet_gzip table.

```
SHOW TBLPROPERTIES fformats.pagecounts_parquet_gzip ('totalSize');
```

```
82514126
```

Explore Parquet Files Metadata

1. List output directories on HDFS.

```
hdfs dfs -ls -R /workshops/fformats/lab01/parquet

hdfs dfs -ls -R /workshops/fformats/lab01/parquet_snappy

hdfs dfs -ls -R /workshops/fformats/lab01/parquet_gzip
```

2. Get the size on HDFS for each output directory.

```
hdfs dfs -du -h -s /workshops/fformats/lab01/parquet

hdfs dfs -du -h -s /workshops/fformats/lab01/parquet_snappy

hdfs dfs -du -h -s /workshops/fformats/lab01/parquet_gzip
```

3. Show metadata of a Parquet bucket.

```
hadoop jar parquet-tools meta
/workshops/fformats/lab01/parquet_snappy/000000_0
```

```
hadoop jar parquet-tools meta
/workshops/fformats/lab01/parquet_gzip/000000_0
```

4. Extract the Parquet file schema.

```
hadoop jar parquet-tools schema
/workshops/fformats/lab01/parquet_snappyp/000000_0
```

```
hadoop jar parquet-tools schema
/workshops/fformats/lab01/parquet_gzip/000000_0
```

```
message hive_schema {
  optional binary projectcode (STRING);
  optional binary pagename (STRING);
  optional binary pageviews (STRING);
  optional binary bytes (STRING);
}
```

Workshop 6

Part 2-d

File Formats: ORC Format

General Instructions:

Apache ORC (Optimized Row Columnar) is a column-oriented data storage format. It provides a highly efficient way to store Hive data. It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data. It is commonly used by most of the data processing frameworks such as Apache Spark, Apache Flink and Apache Hadoop.

The ORC file format supports ACID transactions when working with Hive. It stores collections of rows in a single file, in a columnar format within the file. This enables parallel processing of row collections across a cluster. Due to the columnar layout, each file is optimal for compression, enabling skipping of data and columns to reduce read and decompression loads.

Online resources:

https://orc.apache.org/

Note: *This workshop is divided into 5 parts and should be accomplished in the same order.*

Exercise 2-d: Converting Data to ORC.

In this part of the exercise, you will repeat the steps of the previous parts to convert the initial dataset to ORC format.

In this activity you will:

- Create an ORC table with no compression.
- Create two ORC tables and use two compression codecs: `Snappy` and `ZLib`
- Compare the properties of these tables to the initial dataset.
- Explore output metadata.

ORC Table with No Compression

1. Create a new Zeppelin note and choose the Hive interpreter (`%hive`).
2. By default, Output Compression is disabled in Hive, keep this setting.

```
set hive.exec.compress.output=false;
```

3. Create a new User Managed table named pagecounts_orc and populated from the initial table.

```
create table fformats.pagecounts_orc stored as orc LOCATION
'/workshops/fformats/lab01/orc' as select * from fformats.pagecounts;
```

4. Compute statistics for the pagecounts_orc table.

```
Analyze table fformats.pagecounts_orc compute statistics;
```

5. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_orc;
```

6. Show the first 10 rows from the table.

```
select * from fformats.pagecounts_orc limit 10;
```

7. Report the size of the pagecounts_orc table.

```
SHOW TBLPROPERTIES fformats.pagecounts_orc ('totalSize');
```

89219294

ORC Table with Snappy Compression

8. Enable Output Compression and set the compression codec to Snappy.

```
set hive.exec.compress.output=true;
```

9. Create a new User Managed table named pagecounts_orc_snappy and populated from the initial table.

```
create table fformats.pagecounts_orc_snappy stored as parquet LOCATION
'/workshops/fformats/lab01/orc_snappy' TBLPROPERTIES
('orc.compress'='SNAPPY') as select * from fformats.pagecounts;
```

10. Compute statistics for the pagecounts_orc_snappy table.

```
Analyze table fformats.pagecounts_orc_snappy compute statistics;
```

11. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_orc_snappy;
```

12. Report the size of the pagecounts_orc_snappy table.

```
SHOW TBLPROPERTIES fformats.pagecounts_orc_snappy ('totalSize');
```

124992645

ORC Table with ZLib Compression

6. Enable Output Compression and set the compression codec to ZLib.

```
set hive.exec.compress.output=true;
```

7. Create a new User Managed table named pagecounts_orc_zlib and populated from the initial table.

```
create table fformats.pagecounts_orc_zlib stored as orc LOCATION
'/workshops/fformats/lab01/parquet_orc_zlib ' TBLPROPERTIES
('parquet.compression'='ZLIB') as select * from fformats.pagecounts;
```

8. Compute statistics for the pagecounts_orc_zlib table.

```
Analyze table fformats.pagecounts_orc_zlib compute statistics;
```

9. View description of the table in a formatted way.

```
describe formatted fformats.pagecounts_orc_zlib;
```

10. Report the size of the pagecounts_orc_zlib table.

```
SHOW TBLPROPERTIES fformats.pagecounts_orc_zlib ('totalSize');
```

89219294

Explore ORC Files Metadata

11. List output directories on HDFS.

```
hdfs dfs -ls -R /workshops/fformats/lab01/orc

hdfs dfs -ls -R /workshops/fformats/lab01/orc_snappy

hdfs dfs -ls -R /workshops/fformats/lab01/orc_zlib
```

12. Get the size on HDFS for each output directory.

```
hdfs dfs -du -h -s /workshops/fformats/lab01/orc

hdfs dfs -du -h -s /workshops/fformats/lab01/orc_snappy

hdfs dfs -du -h -s /workshops/fformats/lab01/orc_zlib
```

13. Show metadata of an ORC bucket.

```
hive --orcfiledump /workshops/fformats/lab01/orc/000000_0

hive --orcfiledump /workshops/fformats/lab01/orc_snappy/000000_0

hive --orcfiledump /workshops/fformats/lab01/orc_zlib/000000_0
```

14. Show metadata using orc-tools (working with files stored on local filesystem).

```
hdfs dfs -get /workshops/fformats/lab01/orc_zlib/000000_0
/home/training/Downloads/orc_zlib_0

orc-tools meta /home/training/Downloads/orc_zlib_0
```

Workshop 6

Part 2-e

File Formats Comparison

General Instructions:

The goal of this comparison is to introduce and explain why you may need to convert your data to Avro, Parquet, or ORC. The right data format is essential to achieving optimal performance and desired business outcomes. The choices and nuances of big data formats can be overwhelming. Increasingly, analysts, data scientists, engineers and business users need to know these formats in order to make decisions and understand workflows.

Note: *This workshop is divided into 5 parts and should be accomplished in the same order.*

Exercise 2-e: Comparing Metrics.

In this part of the exercise, you will report the size of each file format experiment and compare them to each other.

In this activity you will:

- Create a staging table. This table has three columns:
  - format string
  - size int
  - compression string
- Populate with sizes from the experiments.
- Create a bar chart graph to visualize data.

Staging Table with Metrics

1. Create a new Zeppelin note and choose the Trino interpreter (`%trino`).
2. Create a new `schema` using the `memory` connector.

```
drop table if exists memory.fformats.formats;

create schema if not exists memory.fformats;

create table memory.fformats.formats (format varchar(25), size int,
compression varchar(25));
```

3. Populate the table with the reported values from the experiments.

```
insert into memory.fformats.formats values('txt', 340304295, 'no');
insert into memory.fformats.formats values('avro', 366391926, 'no');
insert into memory.fformats.formats values('avro_snappy', 147242278, 'yes');
insert into memory.fformats.formats values('avro_deflate', 94318706, 'yes');
insert into memory.fformats.formats values('parquet', 325611113, 'no');
insert into memory.fformats.formats values('parquet_snappy', 129128553, 'yes');
insert into memory.fformats.formats values('parquet_gzip', 82514126, 'yes');
insert into memory.fformats.formats values('orc', 89219294, 'yes');
insert into memory.fformats.formats values('orc_snappy', 124992645, 'yes');
insert into memory.fformats.formats values('orc_zlib', 89219294, 'yes');
```

4. Show all the rows from the table.

```
select * from memory.fformats.formats order by size desc;
```