Taller de Preparación – Certificación Apache Spark Cloudera 2021

Este taller forma parte del programa Data Engineering dictado por Datahack, y del curso de Apache Spark

Iniciamos los servicios de Hadoop y YARN:

- `HDFS - start-dfs.sh`
- `YARN - start-yarn.sh`

Copiamos los ficheros que usaremos en nuestra práctica

```
hdfs dfs -put /data/retail_db /public
hdfs dfs -ls /public/retail_db
```


Ejercicio 1:

Origen:

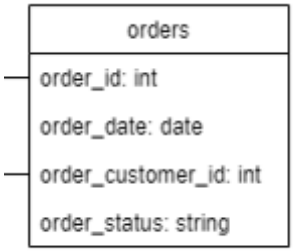
Convertir un archivo csv en avro y almacenarlo en una ruta de HDFS

hdfs: **/public/retail_db/orders/**

Resultado:

Almacena la información en: **/user/vagrant/lab1/pregunta1/resultado**

Con el siguiente esquema:



1. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
import org.apache.spark.sql.types._

val customSchema = StructType(Array(
  StructField("order_id", IntegerType, true),
  StructField("order_date", DateType, true),
  StructField("order_customer_id", IntegerType, true),
  StructField("order_status", StringType, true))
)

val orders = spark.read.format("csv").option("inferSchema",
"true").schema(customSchema).load("/public/retail_db/orders")

orders.write.format("avro").save("/user/vagrant/lab1/pregunta1/orde
rs_avro")

                                                    http://www.datahack.la
```

```
val validate =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/result
ado/part-00000-cdedfeca-66a3-43de-a6cf-318d13780da8-c000.avro")

validate.printSchema()
validate.count()
validate.show()

                                                    http://www.datahack.la
```

1. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
from pyspark.sql.types import *

customSchema = StructType([
   StructField("order_id", IntegerType(), True),
   StructField("order_date", DateType(), True),
   StructField("order_customer_id", IntegerType(), True),
   StructField("order_status", StringType(), True)
   ])

orders = spark.read.option("inferSchema",
"true").schema(customSchema).csv("/public/retail_db/orders")

orders.write.format("avro").save("/user/vagrant/lab1/pregunta1/orde
rs_avro")
```

http://www.datahack.la

```
validate =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/result
ado ")
validate.printSchema()
validate.count()
validate.show()
```

http://www.datahack.la

Ejercicio 2:

Origen:

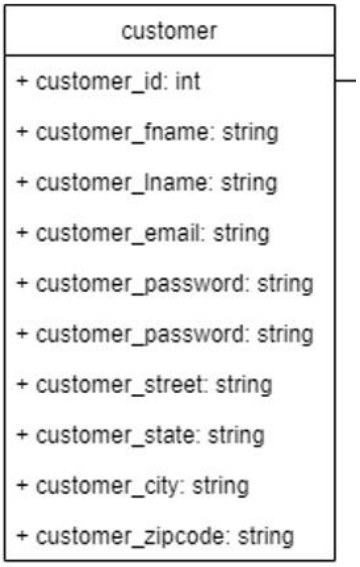
Convertir un archivo csv en parquet y almacenarlo en una ruta de HDFS

hdfs: **/public/retail_db/customers/part-00000**

Resultado:

Almacena la información en: **/user/vagrant/lab1/pregunta2/resultado**

Con el siguiente esquema:



1. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val customerSchema = StructType(Array(
  StructField("customer_id", IntegerType, true),
  StructField("customer_fname", StringType, true),
  StructField("customer_lname", StringType, true),
  StructField("customer_email", StringType, true),
  StructField("customer_password", StringType, true),
  StructField("customer_street", StringType, true),
  StructField("customer_state", StringType, true),
  StructField("customer_city", StringType, true),
  StructField("customer_zipcode", StringType, true)
  )
)
```

```
val customer = spark.read.format("csv").option("inferSchema",
"true").schema(customerSchema).load("/public/retail_db/customers")

customer.write.format("parquet").save("/user/vagrant/lab1/pregunta2
/customer")
```


2. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
customerSchema = StructType([
   StructField("customer_id", IntegerType(), True),
   StructField("customer_fname", StringType(), True),
   StructField("customer_lname", StringType(), True),
   StructField("customer_email", StringType(), True),
   StructField("customer_password", StringType(), True),
   StructField("customer_street", StringType(), True),
   StructField("customer_state", StringType(), True),
   StructField("customer_city", StringType(), True),
   StructField("customer_zipcode", StringType(), True)
   ])

customer =
spark.read.schema(customerSchema).csv("/public/retail_db/customers/
part-00000")

customer.write.format("parquet").save("/user/vagrant/lab1/pregunta2
/customer")
```

Ejercicio 3:

Origen:

Leer la tabla orders en formato avro y aplicar una comprimir en gzip

hdfs: **/user/vagrant/lab1/pregunta1/resultado/part-00000-cdedfeca-66a3-43de-a6cf-318d13780da8-c000.avro**

Resultado:

Almacena la información en: **/user/vagrant/lab1/pregunta3/resultado**

Con el siguiente esquema: Order_id, order_status

3. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val orders_avro =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/orders
_avro/part-00000-26e036ad-b866-4723-bab3-a450da706f5a-c000.avro")

orders_avro.select("order_id","order_status").write.option("compres
sion", "gzip").parquet("/user/vagrant/lab1/pregunta3/resultado")
```


3. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
orders_avro =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/orders
_avro/part-00000-26e036ad-b866-4723-bab3-a450da706f5a-c000.avro")

orders_avro.select("order_id","order_status").write.option("compres
sion", "gzip").parquet("/user/vagrant/lab1/pregunta3/resultado")
```

Ejercicio 4:

Origen:

Obtener los clientes que realizaron más de 4 compras

customer: **`/user/vagrant/lab1/pregunta2/customer`**

orders: **`/user/vagrant/lab1/pregunta1/orders_avro/part-00000-26e036ad-b866-4723-bab3-a450da706f5a-c000.avro`**

Resultado:

Almacena la información en formato JSON: **/user/vagrant/lab1/pregunta4/resultado**

Con el siguiente esquema y deber ser en orden descendente:

customer_id, customer_fname, customer_lname, cant

4. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/customer")

customer.createOrReplaceTempView("customer")

val orders_avro =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/orders_avro/part-00000-26e036ad-b866-4723-bab3-a450da706f5a-c000.avro")

orders_avro.createOrReplaceTempView("orders_avro")
```

4. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/cus
tomer")
customer.createOrReplaceTempView("customer")

orders_avro =
spark.read.format("avro").load("/user/vagrant/lab1/pregunta1/orders
_avro/part-00000-26e036ad-b866-4723-bab3-a450da706f5a-c000.avro")

orders_avro.createOrReplaceTempView("orders_avro")

result = spark.sql("select customer_id, customer_fname,
customer_lname ,count(*) as cant from customer a inner join
orders_avro b on a.customer_id = order_customer_id  group  by
customer_id, customer_lname, customer_fname having count(*) > 5 ")

result.sort(result.cant.desc()).write.format("json").save("/user/va
grant/lab1/pregunta4_python/resultado")
```

Ejercicio 5:

Origen:

Obtener el máximo precio de los productos.

products: **`/public/retail_db/products/part-00000`**

Resultado:

Almacena la información en formato text, comprimido en gzip y solo deberá ser un solo file.

/user/vagrant/lab1/pregunta5/resultado

El archivo deberá ser almacenado de la siguiente manera

Product id | product_price

145787 | 100.0

5. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val ProductSchema = StructType(Array(
  StructField("product_id", IntegerType, true),
  StructField("product_category_id", IntegerType, true),
  StructField("product_name", StringType, true),
  StructField("product_description", StringType, true),
  StructField("product_price", FloatType, true),
  StructField("product_image", StringType, true)
  )
)

val product = spark.read.format("csv").option("inferSchema",
"true").schema(ProductSchema).load("/public/retail_db/products/part
-00000")

product.createOrReplaceTempView("product")

val result =spark.sql("select product_id, max(product_price) as
max_price from product group by product_id")

result.createOrReplaceTempView("result")

val result2 =spark.sql("select concat(product_id, '|', max_price)
as data from result ")
```

```
result2.repartition(1).write.option("compression","gzip").format("t
ext").save("/user/vagrant/lab1/pregunta5/resultado")
```


5. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
ProductSchema = StructType([
   StructField("product_id", IntegerType(), True),
   StructField("product_category_id", IntegerType(), True),
   StructField("product_name", StringType(), True),
   StructField("product_description", StringType(), True),
   StructField("product_price", FloatType(), True),
   StructField("product_image", StringType(), True)
   ])

product = spark.read.format("csv").option("inferSchema",
"true").schema(ProductSchema).load("/public/retail_db/products/part
-00000")

product.createOrReplaceTempView("product")

result =spark.sql("select product_id, max(product_price) as
max_price from product group by product_id")

result.createOrReplaceTempView("result")

result2 =spark.sql("select concat(product_id, '|', max_price) as
data from result ")

result2.repartition(1).write.option("compression","gzip").format("t
ext").save("/user/vagrant/lab1/pregunta5/resultado")
```

Ejercicio 6:

Origen:

Obtener los clientes

customer: **`/user/vagrant/lab1/pregunta2/customer`**

Resultado:

Almacena la información en formato text, deberá estar comprimido en bzip2 y con un delimitador “/t”

Concatenar los valores del first_name (3 caracteres) y last_name

/user/vagrant/lab1/pregunta6/resultado

El archivo deberá ser almacenado de la siguiente manera

Customer_id name, customer_street

6. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/cus
tomer")

customer.createOrReplaceTempView("customer")

val result = spark.sql("select customer_id,
concat(substring(customer_fname,1,3),' ', customer_lname) as name ,
customer_street from customer")

result.map(x =>
x.mkString("\t")).write.option("compression","bzip2").format("text"
).save("/user/vagrant/lab1/pregunta6/resultado")
```

6. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/cus
tomer")

customer.createOrReplaceTempView("customer")

val result = spark.sql("select customer_id,
concat(substring(customer_fname,1,3),' ', customer_lname) as name ,
customer_street from customer")

result.rdd.map(lambda x:
"\t".join(map(str,x))).saveAsTextFile("/user/vagrant/lab1/pregunta6
/resultado","org.apache.hadoop.io.compress.BZip2Codec")
```

Ejercicio 7:

Origen:

Cargar información de productos al metastore:

product: **`/public/retail_db/products/part-00000`**

Resultado:

Almacenar en Metastore table products

7. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val product = spark.read.format("csv").option("inferSchema",
"true").schema(ProductSchema).load("/public/retail_db/products/part
-00000")

product.write.format("hive").saveAsTable("product")
```


7. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
product = spark.read.format("csv").option("inferSchema",
"true").schema(ProductSchema).load("/public/retail_db/products/part
-00000")

product.write.format("hive").saveAsTable("product")
```

Ejercicio 8:

Origen:

Agrupar cantidad de transacciones por mes utilizando el metastore:

orders: **`/public/retail_db/orders/part-00000`**

Resultado:

Almacenar el resultado, en formato parquet sin comprimir en la ruta:

/user/vagrant/lab1/pregunta8/resultado

El archivo deberá ser almacenado con el esquema

Count, month (format YYYYMM)

8. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val orders = spark.read.format("csv").option("inferSchema",
"true").schema(customSchema).load("/public/retail_db/orders/part-
00000")

orders.write.format("hive").saveAsTable("orders")

val result = spark.sql("select count(*) as
count,date_format(order_date,'YYYYMM') as month from orders group
by date_format(order_date, 'YYYYMM')")

result.write.option("compression","uncompressed").format("parquet")
.save("/user/vagrant/lab1/pregunta8/resultado")
```

8. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
orders = spark.read.format("csv").option("inferSchema",
"true").schema(customSchema).load("/public/retail_db/orders/part-
00000")

orders.write.format("hive").saveAsTable("orders")

result = spark.sql("select count(*) as
count,date_format(order_date,'YYYYMM') as month from orders group
by date_format(order_date, 'YYYYMM')")

result.write.option("compression","uncompressed").format("parquet")
.save("/user/vagrant/lab1/pregunta8/resultado")
```

Ejercicio 9:

Origen:

Guardar la información de ítem y transacciones en formato ORC

Order_items: **`/user/vagrant/lab1/pregunta9_2/resultado`**

Resultado:

Almacenar el resultado sin comprimir: **/user/vagrant/lab1/pregunta9/resultado**

9. Respuesta - Spark Scala:

Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val itemsSchema = StructType(Array(
  StructField("order_item_id", IntegerType, true),
  StructField("order_item_order_id", IntegerType, true),
  StructField("order_item_product_id", IntegerType, true),
  StructField("order_item_quantity", IntegerType, true),
  StructField("order_item_subtotal", FloatType, true),
  StructField("order_item_productprice", FloatType, true)
  )
)

val  order_items= spark.read.format("csv").option("inferSchema",
"true").schema(itemsSchema).load("/public/retail_db/order_items/par
t-00000")

order_items.write.format("orc").option("compression","uncompressed"
).save("/user/vagrant/lab1/pregunta9/resultado")
```

9. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
itemsSchema = StructType([
   StructField("order_item_id", IntegerType(), True),
   StructField("order_item_order_id", IntegerType(), True),
   StructField("order_item_product_id", IntegerType(), True),
   StructField("order_item_quantity", IntegerType(), True),
   StructField("order_item_subtotal", FloatType(), True),
   StructField("order_item_productprice", FloatType(), True)
   ])

order_items= spark.read.format("csv").option("inferSchema",
"true").schema(itemsSchema).load("/public/retail_db/order_items/par
t-00000")

order_items.write.format("orc").option("compression","uncompressed"
).save("/user/vagrant/lab1/pregunta9/resultado")
```

Ejercicio 10:

Origen:

Obtener información de los clientes top de transacciones en cantidad y monto de consumo que se ubican en ciudades que empiecen con M.

customer: **/user/vagrant/lab1/pregunta2/customer**

order_items: /user/vagrant/lab1/pregunta9/resultado

orders: metaestore table orders

Resultado:

Almacenar el resultado en formato parquet comprimido en Gzip:

/user/vagrant/lab1/pregunta10/resultado

4. Respuesta - Spark Scala:

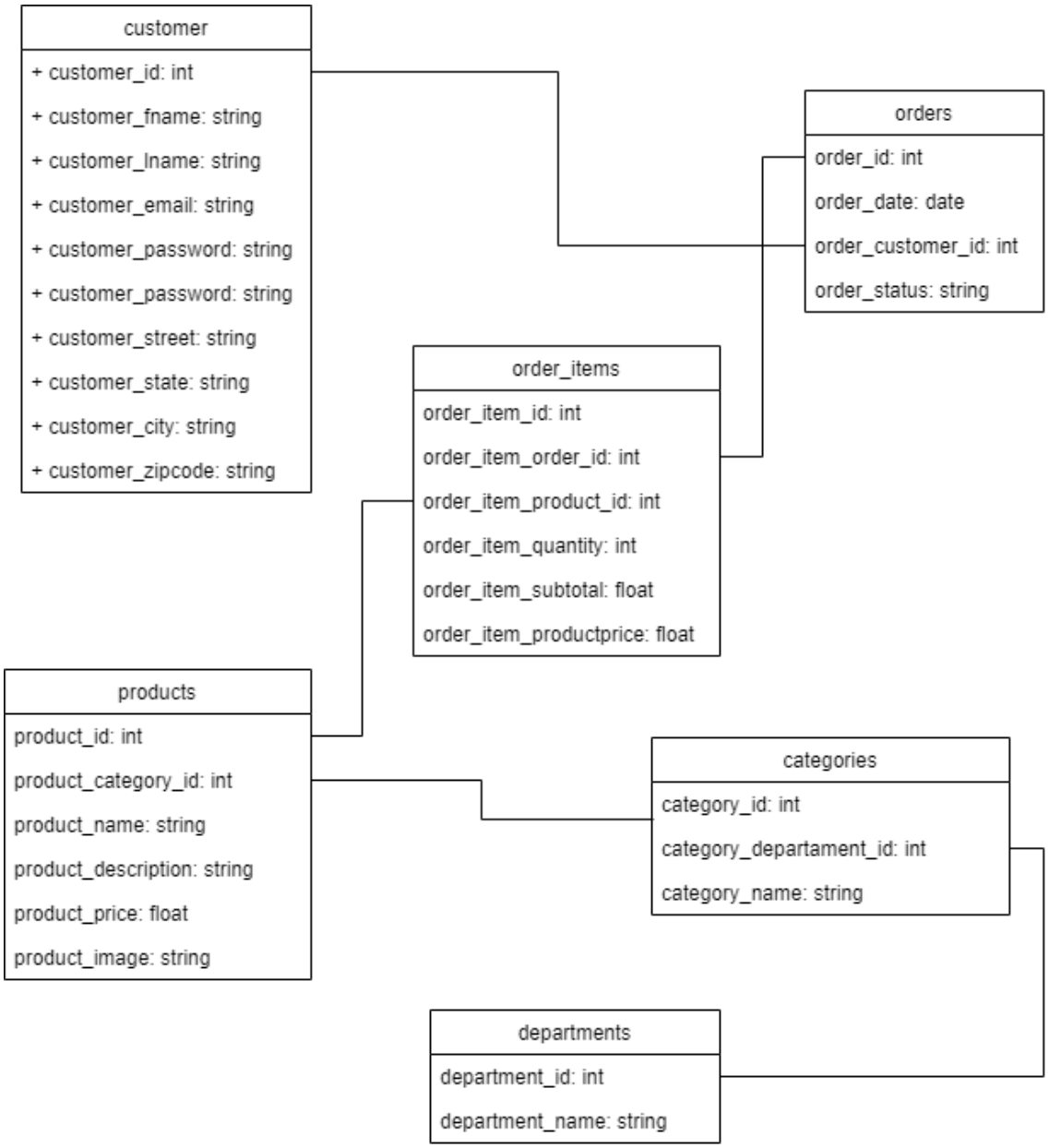
Requerimiento: spark-shell --packages org.apache.spark:spark-avro_2.11:2.4.4

```
val customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/cus
tomer")

customer.createOrReplaceTempView("customer")

val order_items =
spark.read.format("orc").load("/user/vagrant/lab1/pregunta9/resulta
do")

order_items.createOrReplaceTempView("order_items")

val top_customer = spark.sql("select customer_id, customer_fname,
count(*) as cant, sum(order_item_subtotal) as total   from customer
a inner join orders b  on a.customer_id = b.order_customer_id inner
join order_items c on c.order_item_order_id = b.order_id where
customer_city like 'M%' group by customer_id, customer_fname")


top_customer.write.format("parquet").option("compression","gzip").s
ave("/user/vagrant/lab1/pregunta10/resultado")
```

4. Respuesta - Pyspark:

Requerimiento: pyspark --packages org.apache.spark:spark-avro_2.11:2.4.4

```
customer =
spark.read.format("parquet").load("/user/vagrant/lab1/pregunta2/cus
tomer")

customer.createOrReplaceTempView("customer")

order_items =
spark.read.format("orc").load("/user/vagrant/lab1/pregunta9/resulta
do")

order_items.createOrReplaceTempView("order_items")

top_customer = spark.sql("select customer_id, customer_fname,
count(*) as cant, sum(order_item_subtotal) as total   from customer
a inner join orders b  on a.customer_id = b.order_customer_id inner
join order_items c on c.order_item_order_id = b.order_id where
customer_city like 'M%' group by customer_id, customer_fname")

top_customer.write.format("parquet").option("compression","gzip").s
ave("/user/vagrant/lab1/pregunta10/resultado")
```


customer
+ customer_id: int
+ customer_fname: string
+ customer_lname: string
+ customer_email: string
+ customer_password: string
+ customer_password: string
+ customer_street: string
+ customer_state: string
+ customer_city: string
+ customer_zipcode: string
orders
order_id: int
order_date: date
order_customer_id: int
order_status: string
order_items
order_item_id: int
order_item_order_id: int
order_item_product_id: int
order_item_quantity: int
order_item_subtotal: float
order_item_productprice: float
products
product_id: int
product_category_id: int
product_name: string
product_description: string
product_price: float
product_image: string
categories
category_id: int
category_departament_id: int
category_name: string
departments
department_id: int
department_name: string