11. Examen Resuelto Certificación CCA Cloudera Spark and Hadoop Developer

1) sqoop import desde un database a HDFS separado por coma (default)

Para resolver este problema vamos a utilizar la máquina virtual de Cloudera 5.12 que viene preinstalada con una MySQL y con un conjunto de datos almacenados en la base de datos reatil_db.

Vamos a comprobar que disponemos del dataset

```
mysql -uroot -pcloudera
show databases;
use retail_db:
show tables;
```

Vamos a importar la tabla products en el HDFS, separando los campos por comas y dejando en el directorio /user/cloudera/examen/ejercicio1

```
sqoop import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table products \
--fields-terminated-by "," \
--target-dir /user/cloudera/examen \
--as-textfile \
-m 1
```

2) sqoop export desde un Hdfs a una tabla que ya existe de mysql

Vamos a crear una tabla en la base de datos mysql para exportar los datos que acabamos de importar en el ejercicio anterior.

```
CREATE TABLE products_replica
(
     product_id_r int(11),
     product_category_id_r int(11),
     product_name_r varchar(45),
     product_description_r varchar(255),
     product_price_r float,
     product_image_r varchar(255)
)
```

Vamos a utilizar Sqoop para exportar los datos que se encuentran en el directorio /home/cloudera/examen para cargar la tabla de mysql products_replica

```
sqoop export \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table products_replica \
--export-dir /user/cloudera/examen
```

3) creación de una tabla con hive/impala y hacer una select los files estan en hadoop

Abrimos la shell de Hive

```
>hive

CREATE EXTERNAL TABLE products_hive
(
     product_id int,
     product_category_id int,
     product_name string,
     product_description string,
     product_price float,
     product_image string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/cloudera/examen'
```

Comprobar que el resultado de las filas de la tabla es el esperado

```
SELECT  product_id, product_category_id, product_name,
product_description, product_price, product_image
FROM products_hive;
```

4) transformación de una tabla en formato texto a un formado AVRO o parquet

La tabla que hemos creado en el ejercicio anterior esta almacenada como texto así que es la que vamos a utilizar para realizar la transformación a AVRO.

//Esta solución no es suficiente porque debemos transformar los ficheros de texto a ficheros Avro.

```
ALTER TABLE products_hive SET FILEFORMAT AVRO;
```

//Vamos a intentarlo con una CTAS sobre la tabla que tenemos creada en textfile.

```
CREATE TABLE products_avro
STORED AS AVRO
AS
SELECT product_id, product_category_id,product_name,
product_description, product_price, product_image
FROM products_hive
```

Podemos comprobar que se han creado los ficheros correspondientes en el directorio

```
hdfs dfs -ls /user/hive/warehouse/products_avro/
```

//Vamos a crear también la tabla en Parquet

```
CREATE TABLE products_parquet
STORED AS PARQUET
AS
SELECT product_id, product_category_id,product_name,
product_description, product_price, product_image
FROM products_hive
```

Para ver el contenido que tiene el fichero del directorio de la tabla que acabamos de crear:

```
hdfs dfs -cat /user/hive/warehouse/products_parquet/000000_0
```

5) crear una tabla hive / impala con particiones (un campo)

```
CREATE TABLE products_partitioned
(
     product_id int,
     product_name string,
     product_description string,
     product_price float,
     product_image string
)
PARTITIONED BY (product_category_id int)
STORED AS TEXTFILE
```

6) completar un script python spark o scala spark con groupbykey, sc.textfile
Calcular el número total de categorías en la tabla products (he utilizado los ficheros en el directorio HDFS importado previamente con Sqoop)

Scala:

```
spark-shell
val rdd = sc.textFile("/user/cloudera/examen")
```

```
rdd.map(line => line.split(",")).map(fields =>
(fields(1),(fields(0),fields(2),fields(3),fields(4),fields(5)))).g
roupByKey().count()
```

Python

```
pyspark --master local[2]
rdd.textFile("/user/cloudera/examen")

rdd.map(lambda line: line.split(",")).map(lambda fields:
(fields[1],(fields[0],fields[2],fields[3],fields[4],fields[5]))).g
roupByKey().count()
```

7) completar un script python spark o scala spark con groupbykey, sc.textfile
En este ejercicio en vez de utilizar la operación de agrupación groupByKey(), vamos a utilizar reduceByKey()

```
spark-shell
val rdd = sc.textFile("/user/cloudera/examen")

rdd.map(line => line.split(",")).map(fields =>
(fields(1),(fields(0),fields(2),fields(3),fields(4),fields(5)))).r
educeByKey((x,y) => (x)).count()
```

8) completar un script python spark o scala spark con groupbykey, sc.textfile, saveastextfile, groupbykey etc
El siguiente ejercicio vamos a resolver un problema y guardar el resultado en el hdfs como fichero de texto. El problema consiste en leer los datos de la tabla customers de mysql y obtener el estado con mayor número de clientes.

1º-Importar datos desde MySQL al HDFS con Sqoop

```
sqoop import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table customers \
--fields-terminated-by "," \
--target-dir /user/cloudera/customers \
--as-textfile \
-m 1
```

2º - Leer y transformar con Scala

val rdd = sc.textFile("/user/cloudera/customers")

//Mejor que la anterior tenemos que hacer simplemente una tupla con nombre_estado, y 1, para después sumarlas y obtener el estado con más clientes.

```
rdd.map(line => line.split(",")).map(fields =>
(fields(7),1)).reduceByKey((v1,v2) => v1+v2
).collect().sortBy(_._2)
```

9) completar un script python spark o scala spark con join etc

Para este ejercicio vamos a importar dos de las tablas de MySQL, son orders y order_items

```
sqoop import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table orders \
--target-dir /user/cloudera/orders \
--fields-terminated-by  "," \
--as-textfile \
-m 1

sqoop import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table order_items \
--target-dir /user/cloudera/order_items \
--fields-terminated-by  "," \
--as-textfile \
-m 1
```

Vamos a realizar un join entre las dos tablas por el campo que comparten. Para ello vamos a preparar dos RDD [K,V] para que el join sea automático. (Más tarde se pueden calcular cosas mas interesantes como cual es el pedido que mas items tiene asociados)

```
val orders = sc.textFile("/user/cloudera/orders").map(line =>
line.split(",")).map(fields => (fields(0),
(fields(1),fields(2),fields(3))))

val order_items =
sc.textFile("/user/cloudera/order_items").map(line =>
line.split(",")).map(fields => (fields(1),
(fields(0),fields(2),fields(3),fields(4),fields(5))))

orders.join(order_items).groupByKey().take(10)
```

10) Avro, re-escribir un file de cabecera avro para que pueda leer los nuevos datos

Vamos a importar una tabla de la base de datos de MySQL en formato AVRO. Esta tabla va a ser

```
sqoop import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table categories \
--target-dir /user/cloudera/categories \
--fields-terminated-by  "," \
--as-avrodatafile \
-m 1
```

¿Donde esta el fichero que define el esquema AVRO?
Tenemos que traernos el fichero que acabamos de importar a local, y utilizar la herramienta avro

```
hdfs dfs -get /user/cloudera/categories/part-m-00000.avro
avro-tools getschema part-m-00000.avro > schema.avsc

hdfs dfs -mkdir /user/cloudera/categories_schema

hdfs dfs -put schema.avsc
/user/cloudera/categories_schema/schema.avsc
```

Ahora tenemos que crear una tabla en Hive, con el esquema que acabamos de obtener y que apunte al directorio HDFS donde hemos realizado la importación de los datos.

```
CREATE EXTERNAL TABLE categories_avro
STORED AS AVRO
LOCATION '/user/cloudera/categories'
TBLPROPERTIES('avro.schema.url'='/user/cloudera/categories_schema/
schema.avsc');
```

Ahora que ya tenemos nuestros datos cargados como ficheros avro, lo que vamos hacer es modificar la definición del esquema avro que acabamos de crear, tras ello realizaremos una inserción en la tabla para comprobar que todo ha funcionado correctamente.

1. Nos vamos al fichero local schema.avsc y añadimos una nueva columna que sea descripcion.

```
cp schema.avsc schema2.avsc
nano schema2.avsc
```

2. Lo subimos al directorio donde hemos guardado nuestra primera versión del schema avro.

```
hdfs dfs -put schema2.avsc /user/cloudera/categories_schema
hdfs dfs -ls /user/cloudera/categories_schema
```

3. Y ahora modificamos las propiedades de la tabla para que apunte al nuevo esquema

```
ALTER TABLE default.categories_avro SET TBLPROPERTIES
('avro.schema.url' =
'/user/cloudera/categories_schema/schema2.avsc');
```