1- Creación de tablas en formato texto

• 1.1) Crear Base de datos "datos_padron"

```
CREATE DATABASE datos_padron
```

• 1.2) Crear la tabla de datos padron_txt con todos los campos del fichero CSV y cargar los datos mediante el comando LOAD DATA LOCAL INPATH. La tabla tendrá formato texto y tendrá como delimitador de campo el caracter ';' y los campos que en el documento original están encerrados en comillas dobles "" no deben estar envueltos en estos caracteres en la tabla de Hive (es importante indicar esto utilizando el serde de OpenCSV, si no la importación de las variables que hemos indicado como numéricas fracasará ya que al estar envueltos en comillas los toma como strings) y se deberá omitir la cabecera del fichero de datos al crear la tabla.

```
CREATE TABLE padron_txt(
COD_DISTRITO INT,
DESC_DISTRITO STRING,
COD_DIST_BARRIO INT,
DESC_BARRIO STRING,
COD_BARRIO INT,
COD_DIST_SECCION INT,
COD_SECCION INT,
COD_EDAD_INT INT,
EspanolesHombres INT,
EspanolesMujeres INT,
ExtranjerosHombres INT,
ExtranjerosMujeres INT )
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ("separatorChar" = ";", "quoteChar"='"')
STORED AS TEXTFILE TBLPROPERTIES ("skip.header.line.count"="1");
```

• 1.3) Hacer trim sobre los datos para eliminar los espacios innecesarios guardando la tabla resultado como padron_txt_2. (Este apartado se puede hacer creando la tabla con una sentencia CTAS.)

```
CREATE TABLE padron_txt_2 AS SELECT trim(cod_distrito) as cod_distrito, trim(desc_distrito) as desc_distrito, trim(cod_dist_barrio) as cod_dist_barrio,
trim(desc_barrio) as desc_barrio,
trim(cod_barrio) as cod_barrio, trim(cod_dist_seccion) as cod_dist_seccion, trim(cod_seccion) as cod_seccion,
trim(cod_edad_int) as cod_edad_int,
trim(espanoleshombres) as espanoleshombres, trim(espanolesmujeres) as espanolesmujeres,
trim(extranjeroshombres) as extranjeroshombres, trim(extranjerosmujeres) as extranjerosmujeres

FROM padron_txt
```

• 1.4) Investigar y entender la diferencia de incluir la palabra LOCAL en el comando LOAD DATA.

LOAD DATA command is used to load data into a hive table.

- LOCAL: Input file is on the local file system.
- **NO** LOCAL: Input file is in HDFS.

• 1.5) En este momento te habrás dado cuenta de un aspecto importante, los datos nulos de nuestras tablas vienen representados por un espacio vacío y no por un identificador de nulos comprensible para la tabla. Esto puede ser un problema para el tratamiento posterior de los datos. Podrías solucionar esto creando una nueva tabla utilizando sentencias case cuando sustituyan espacios en blanco por 0. Para esto primero comprobaremos que solo hay espacios en blanco en las variables numéricas correspondientes a las últimas 4 variables de nuestra tabla (podemos hacerlo con alguna sentencia de HiveQL) y luego aplicaremos las sentencias case when para sustituir por 0 los espacios en blanco. (Pista: es útil darse cuenta de que un espacio vacío es un campo con longitud 0). Haz esto solo para la tabla padron_txt.

```
CREATE TABLE padron_txt_nulls AS SELECT trim(cod_distrito) as cod_distrito, trim(desc_distrito) as desc_distrito, trim(cod_dist_barrio) as cod_dist_barrio,
trim(desc_barrio) as desc_barrio,
trim(cod_barrio) as cod_barrio, trim(cod_dist_seccion) as cod_dist_seccion, trim(cod_seccion) as cod_seccion,
trim(cod_edad_int) as cod_edad_int,

CASE WHEN length(espanoleshombres)=0 THEN '0' ELSE espanoleshombres
END AS espanoleshombres,

CASE WHEN length(espanolesmujeres)=0 THEN '0' ELSE espanolesmujeres
END AS espanolesmujeres,

CASE WHEN length(extranjeroshombres)=0 THEN '0' ELSE extranjeroshombres
END AS extranjeroshombres,

CASE WHEN length(extranjerosmujeres)=0 THEN '0' ELSE extranjerosmujeres
END AS extranjerosmujeres

FROM padron_txt
```

• 1.6) Una manera tremendamente potente de solucionar todos los problemas previos (tanto las comillas como los campos vacíos que no son catalogados como null y los espacios innecesarios) es utilizar expresiones regulares (regex) que nos proporciona OpenCSV. Para ello utilizamos :

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe' WITH SERDEPROPERTIES ('input.regex'='XXXXXXX') Donde XXXXXX representa una expresión regular que debes completar y que identifique el formato exacto con el que debemos interpretar cada una de las filas de nuestro CSV de entrada. Para ello puede ser útil el portal "regex101". Utiliza este método para crear de nuevo la tabla padron_txt_2. Una vez finalizados todos estos apartados deberíamos tener una tabla padron_txt que conserve los espacios innecesarios, no tenga comillas envolviendo los campos y los campos nulos sean tratados como valor 0 y otra tabla padron_txt_2 sin espacios innecesarios, sin comillas envolviendo los campos y con los campos nulos como valor 0. Idealmente esta tabla ha sido creada con las regex de OpenCSV.

Regex

"\"([0-9])\";\"([A-Z]*[\\s]*)\";\"([0-9]*)\";\"([A-Z]*[\\s]*)\";\"([0-9]*)\";\"([0-9]*)\";\"([0-9]*)\";\"([0-9]*)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\""*

```
CREATE TABLE padron_txt_2(
COD_DISTRITO STRING,
DESC_DISTRITO STRING,
COD_DIST_BARRIO STRING,
DESC_BARRIO STRING,
COD_BARRIO STRING,
COD_DIST_SECCION STRING,
COD_SECCION STRING,
COD_EDAD_INT STRING,
EspanolesHombres STRING,
EspanolesMujeres STRING,
ExtranjerosHombres STRING,
ExtranjerosMujeres STRING)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES
("input.regex"="\"([0-9]*)\";\"([A-Z]*[\\s]*)\";\"([0-9]*)\";\"([A-Z]*[\\s]*)\";\"([0-9]*)\";\"([0-9]
*)\";\"([0-9]*)\";\"([0-9]*)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\";\"([0-9]*|\\s)\"")
STORED AS TEXTFILE TBLPROPERTIES ("skip.header.line.count"="1")
```

2- Investigamos el formato columnar parquet

• 2.1) ¿Qué es CTAS?

CREATE TABLE AS SELECT

• 2.2) Crear tabla Hive padron_parquet (cuyos datos serán almacenados en el formato columnar parquet) a través de la tabla padron_txt mediante un CTAS.

```
CREATE TABLE padron_parquet STORED AS PARQUET AS SELECT cod_distrito as cod_distrito, desc_distrito as desc_distrito, cod_dist_barrio as cod_dist_barrio,
desc_barrio as desc_barrio,
cod_barrio as cod_barrio, cod_dist_seccion as cod_dist_seccion, cod_seccion as cod_seccion,
cod_edad_int as cod_edad_int,
espanoleshombres as espanoleshombres,espanolesmujeres as espanolesmujeres,
extranjeroshombres as extranjeroshombres, extranjerosmujeres as extranjerosmujeres
FROM padron_txt
```

• 2.3) Crear tabla Hive padron_parquet_2 a través de la tabla padron_txt_2 mediante un CTAS. En este punto deberíamos tener 4 tablas, 2 en txt (padron_txt y padron_txt_2, la primera con espacios innecesarios y la segunda sin espacios innecesarios) y otras dos tablas en formato parquet (padron_parquet y padron_parquet_2, la primera con espacios y la segunda sin ellos).

```
CREATE TABLE padron_parquet_2 STORED AS PARQUET AS SELECT cod_distrito as cod_distrito, desc_distrito as desc_distrito, cod_dist_barrio as cod_dist_barrio,
desc_barrio as desc_barrio,
cod_barrio as cod_barrio, cod_dist_seccion as cod_dist_seccion, cod_seccion as cod_seccion,
cod_edad_int as cod_edad_int,
espanoleshombres as espanoleshombres,espanolesmujeres as espanolesmujeres,
extranjeroshombres as extranjeroshombres, extranjerosmujeres as extranjerosmujeres
FROM padron_txt_2
```

• 2.5) Investigar en qué consiste el formato columnar parquet y las ventajas de trabajar con este tipo de formatos.

Parquet – Databricks

• 2.6) Comparar el tamaño de los ficheros de los datos de las tablas padron_txt (txt), padron_txt_2 (txt pero no incluye los espacios innecesarios), padron_parquet y padron_parquet_2 (alojados en hdfs cuya ruta se puede obtener de la propiedad location de cada tabla por ejemplo haciendo "show create table").

padron_parquet

```
TBLPROPERTIES (
'COLUMN_STATS_ACCURATE'='true',
'numFiles'='1',
'numRows'='238918',
'rawDataSize'='2867016',
'totalSize'='877342',
'transient_lastDdlTime'='1666198838')
```

padron_parquet_2

```
TBLPROPERTIES (
'COLUMN_STATS_ACCURATE'='true',
'numFiles'='1',
'numRows'='238918',
'rawDataSize'='2867016',
'totalSize'='379537',
'transient_lastDdlTime'='1666199326')
```

3- Juguemos con Impala

• 3.1) ¿Qué es Impala?

Impala is a MPP (Massive Parallel Processing) SQL query engine for processing huge volumes of data that is stored in a Hadoop cluster. Provides the fastest way to access data that is stored in Hadoop Distributed File System.

• 3.2) ¿En qué se diferencia de Hive?

Latencia	Hive se basa en MapReduce de Hadoop y, por lo tanto, tiene una latencia más alta cuando se trata de procesar consultas. Hive fue construido principalmente para la sofisticación y no para la velocidad.	Impala fue construido principalmente para la velocidad y, por lo tanto, tiene una latencia muy baja.
Rendimiento	Entre los tres, Hive tiene el rendimiento más bajo.	Entre los tres, Impala tiene el mayor rendimiento.

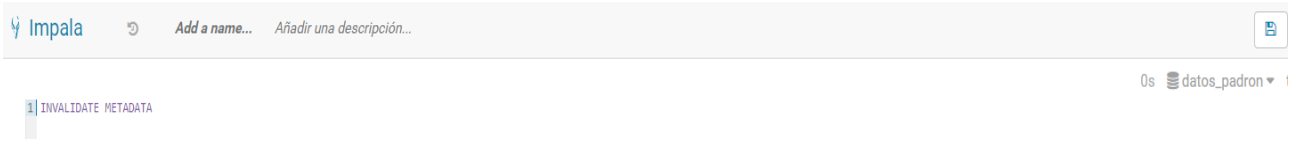
Casos de uso	Hive es ideal para situaciones en las que se requiere compatibilidad multiusuario y con frecuencia se realizan consultas complejas que deben abarcar varias bases de datos.	Impala es más adecuado para cargas de trabajo interactivas empresariales donde se requiere una baja latencia y las consultas deben ser interactivas.
Tolerancia a fallos	Hive puede recuperarse de errores a mitad de consulta.	Impala no es tolerante a fallos. Si la ejecución de la consulta falla a mitad de la consulta, la consulta debe ejecutarse de nuevo.
Complejidad de las consultas	Hive está diseñado para manejar consultas de ejecución prolongada que requieren múltiples transformaciones y uniones.	Impala está diseñado para manejar consultas más cortas en grandes conjuntos de datos, pero debido a su baja latencia, es ideal para la computación interactiva.

• 3.3) Comando INVALIDATE METADATA, ¿en qué consiste?

Se utiliza para actualizar los metadatos de toda la base de datos o de una tabla. Primero borra la memoria caché de la tabla y luego vuelve a cargar todos los datos de la tienda de metadatos y la memoria caché.

INVALIDATE METADATA [[db_name.]table_name]

• 3.4) Hacer invalidate metadata en Impala de la base de datos datos_padron.



• 3.5) Calcular el total de EspanolesHombres, espanolesMujeres, ExtranjerosHombres y ExtranjerosMujeres agrupado por DESC_DISTRITO y DESC_BARRIO.

• 3.6) Llevar a cabo las consultas en Hive en las tablas padron_txt_2 y padron_parquet_2 (No deberían incluir espacios innecesarios). ¿Alguna conclusión?

- SELECT desc_barrio,desc_distrito,count(espanoleshombres) as espanoleshombres,count(espanolesmujeres) as espanolesmujeres,count(extranjeroshombres) as extranjeroshombres,count(extranjerosmujeres) as extranjerosmujeres

  FROM padron_txt_2

  GROUP BY desc_barrio,desc_distrito

44.35 s

- SELECT desc_barrio,desc_distrito,count(espanoleshombres) as espanoleshombres,count(espanolesmujeres) as espanolesmujeres,count(extranjeroshombres) as extranjeroshombres,count(extranjerosmujeres) as extranjerosmujeres

  FROM padron_parquet_2

  GROUP BY desc_barrio,desc_distrito

43.85 s

Es más rápido realizar la consulta en la tabla guardada en formato columnar parquet.

• 3.7) Llevar a cabo la misma consulta sobre las mismas tablas en Impala. ¿Alguna conclusión?

Hive: 44.35 s , 43.85 s
Impala: 8.77 s , 2.80 s

Impala es mucho más rápido.

• 3.8) ¿Se percibe alguna diferencia de rendimiento entre Hive e Impala?

Impala es mucho más rápido al menos para las consultas realizadas.

4- Sobre tablas particionadas

• 4.1) Crear tabla (Hive) padron_particionado particionada por campos DESC_DISTRITO y DESC_BARRIO cuyos datos estén en formato parquet.

```
CREATE TABLE padron_particionado(
COD_DISTRITO STRING,
COD_DIST_BARRIO STRING,
COD_BARRIO STRING,
COD_DIST_SECCION STRING,
COD_SECCION STRING,
COD_EDAD_INT STRING,
EspanolesHombres STRING,
EspanolesMujeres STRING,
ExtranjerosHombres STRING,
ExtranjerosMujeres STRING)

PARTITIONED BY(desc_distrito STRING,desc_barrio STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET TBLPROPERTIES ("skip.header.line.count"="1");
```

• 4.2) Insertar datos (en cada partición) dinámicamente (con Hive) en la tabla recién creada a partir de un select de la tabla padron_parquet_2.

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
INSERT OVERWRITE TABLE padron_particionado_bueno PARTITION (desc_barrio,desc_distrito)
SELECT *
FROM padron_parquet_2;
```

• 4.3) Hacer invalidate metadata en Impala de la base de datos padron_particionado.

```
INVALIDATE METADATA datos_padron.padron_particionado
```

• 4.4) Calcular el total de EspanolesHombres, EspanolesMujeres, ExtranjerosHombres y ExtranjerosMujeres agrupado por DESC_DISTRITO y DESC_BARRIO para los distritos CENTRO, LATINA, CHAMARTIN, TETUAN, VICALVARO y BARAJAS.

SELECT desc_barrio,desc_distrito,count(espanoleshombres) as espanoleshombres,count(espanolesmujeres) as espanolesmujeres, count(extranjeroshombres) as extranjeroshombres, count(extranjerosmujeres) as extranjerosmujeres

FROM padron_txt

WHERE desc_distrito='CENTRO' OR desc_distrito='LATINA' OR desc_distrito='CHAMARTIN' OR desc_distrito='TETUAN' OR desc_distrito='VICALVARO' OR desc_distrito='BARAJAS'

GROUP BY desc_distrito,desc_barrio

• 4.5) Llevar a cabo la consulta en Hive en las tablas padron_parquet y padron_partitionado. ¿Alguna conclusión?

Es mucho más lento en la ejecución de la consulta.

• 4.6) Llevar a cabo la consulta en Impala en las tablas padron_parquet y padron_particionado. ¿Alguna conclusión?

Es más rápido que las consultas realizadas en Hive.

5- Trabajando con tablas en HDFS. A continuación vamos a hacer una inspección de las tablas, tanto externas (no gestionadas) como internas (gestionadas). Este apartado se hará si se tiene acceso y conocimiento previo sobre cómo insertar datos en HDFS.

• 5.3) Crear un directorio en HDFS con un nombre a placer, por ejemplo, /test. Si estás en una máquina Cloudera tienes que asegurarte de que el servicio HDFS está activo ya que puede no iniciarse al encender la máquina (puedes hacerlo desde el Cloudera Manager). A su vez, en las máquinas Cloudera es posible (dependiendo de si usamos Hive desde consola o desde Hue) que no tengamos permisos para crear directorios en HDFS salvo en el directorio /user/cloudera.

```
[cloudera@quickstart ~]$ hadoop fs -mkdir /home/cloudera/test
```

• 5.4) Mueve tu fichero datos1 al directorio que has creado en HDFS con un comando desde consola.

```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/datos1 /home/cloudera/test
```

```
[cloudera@quickstart ~]$ hadoop fs -copyFromLocal /home/cloudera/Desktop/datos1
/home/cloudera/test
```

• 5.5) Desde Hive, crea una nueva database por ejemplo con el nombre numeros. Crea una tabla que no sea externa y sin argumento location con tres columnas numéricas, campos separados por coma y delimitada por filas. La llamaremos por ejemplo numeros_tbl.

```
hive> create database numeros;
```

```
hive> describe tabla_numeros;
OK
n1                      int
n2                      int
n3                      int
```

```
hive> create table tabla_numeros(n1 INT,n2 INT, n3 INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 0.165 seconds
```

• 5.6) Carga los datos de nuestro fichero de texto datos1 almacenado en HDFS en la tabla de Hive. Consulta la localización donde estaban anteriormente los datos almacenados. ¿Siguen estando ahí? ¿Dónde están?. Borra la tabla, ¿qué ocurre con los datos almacenados en HDFS?

```
hive> load data local inpath '/home/cloudera/Desktop/datos1' INTO TABLE tabla_numeros;
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /home/cloudera/test
Found 1 items
-rw-r--r--   1 cloudera supergroup         18 2022-10-19 23:39 /home/cloudera/test/datos1
```

• 5.7) Vuelve a mover el fichero de texto datos1 desde el almacenamiento local al directorio anterior en HDFS.

```
[cloudera@quickstart ~]$ hadoop fs -copyFromLocal '/home/cloudera/Desktop/datos1' '/home/cloudera/test'
```

• 5.8) Desde Hive, crea una tabla externa sin el argumento location. Y carga datos1 (desde HDFS) en ella. ¿A dónde han ido los datos en HDFS? Borra la tabla ¿Qué ocurre con los datos en hdfs?

```
hive> create external table if not exists external_table(n1 INT,n2 INT,n3 INT) ROW FORMAT DELIMITED FIELDS
 TERMINATED BY ',';
```

```
hive> drop table external_table;
OK
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /home/cloudera/test
Found 1 items
-rw-r--r--   1 cloudera supergroup         18 2022-10-19 23:39 /home/cloudera/test/datos1
```

• 5.9) Borra el fichero datos1 del directorio en el que estén. Vuelve a insertarlos en el directorio que creamos inicialmente (/test). Vuelve a crear la tabla numeros desde hive pero ahora de manera externa y con un argumento location que haga referencia al directorio donde los hayas situado en HDFS (/test). No cargues los datos de ninguna manera explícita. Haz una consulta sobre la tabla que acabamos de crear que muestre todos los registros. ¿Tiene algún contenido?

```
hive> create external table if not exists external_table_numeros(n1 INT,n2 INT,n3 INT) ROW FORMAT DELIMITE
D FIELDS TERMINATED BY ',' LOCATION '/home/cloudera/test';
OK
```

```
hive> select * from external_table_numeros;
OK
1       2       3
4       5       6
7       8       9
```

• 5.10) Inserta el fichero de datos creado al principio, "datos2" en el mismo directorio de HDFS que "datos1". Vuelve a hacer la consulta anterior sobre la misma tabla. ¿Qué salida muestra?

```
hive> WARN: The method class org.apache.commons.logging.impl.SLF4JLogFactory#release() was invoked.
WARN: Please see http://www.slf4j.org/codes.html#release for an explanation.
[cloudera@quickstart ~]$ hadoop fs -copyFromLocal '/home/cloudera/Desktop/datos2' '/home/cloudera/test'
[cloudera@quickstart ~]$ hadoop fs -ls /home/cloudera/test
Found 2 items
-rw-r--r--   1 cloudera supergroup         18 2022-10-19 23:39 /home/cloudera/test/datos1
-rw-r--r--   1 cloudera supergroup         24 2022-10-20 00:25 /home/cloudera/test/datos2
```

```
hive> select * from external_table_numeros;
OK
1       2       3
4       5       6
7       8       9
14      5       1
44      16      55
11      47      3
```