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,
  FX_CARGA TIMESTAMP,
  FX_DATOS_INI STRING,
  FX_DATOS_FIN STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  'separatorChar' = ';',
  'quoteChar' = ""
)
STORED AS TEXTFILE
TBLPROPERTIES ('skip.header.line.count'='1');


LOAD DATA LOCAL INPATH '/ejercicios/estadisticas202402.csv' INTO TABLE padron_txt;
```

• 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
  CAST(COD_DISTRITO AS INT),
  TRIM(DESC_DISTRITO) AS DESC_DISTRITO,
  CAST(COD_DIST_BARRIO AS INT),
  TRIM(DESC_BARRIO) AS DESC_BARRIO,
  CAST(COD_BARRIO AS INT),
  CAST(COD_DIST_SECCION AS INT),
  CAST(COD_SECCION AS INT),
  CAST(TRIM(COD_EDAD_INT) AS INT) AS COD_EDAD_INT,
  CAST(TRIM(ESPANOLESHOMBRES) AS INT) AS ESPANOLESHOMBRES,
  CAST(TRIM(ESPANOLESMUJERES) AS INT) AS ESPANOLESMUJERES,
  CAST(TRIM(EXTRANJEROSHOMBRES) AS INT) AS EXTRANJEROSHOMBRES,
  CAST(TRIM(EXTRANJEROSMUJERES) AS INT) AS EXTRANJEROSMUJERES,
  CAST(FX_CARGA AS TIMESTAMP),
  TRIM(FX_DATOS_INI) AS FX_DATOS_INI,
  TRIM(FX_DATOS_FIN) AS FX_DATOS_FIN
FROM padron_txt;
```

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

El uso del LOCAL hace que podamos coger directamente los archivos de nuestro sistema sin tenerlos HDFS antes, aunque posteriormente al usarlo se suben a HDFS automaticamente.

• 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 utiliando sentencias case when que 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.

```
INSERT OVERWRITE TABLE padron_txt_2
SELECT
  COD_DISTRITO,
  DESC_DISTRITO,
  COD_DIST_BARRIO,
  DESC_BARRIO,
  COD_BARRIO,
  COD_DIST_SECCION,
  COD_SECCION,
   CAST(CASE WHEN LENGTH(COD_EDAD_INT) = 0 THEN 0 ELSE COD_EDAD_INT
END AS INT) AS COD_EDAD_INT,
     CAST(CASE WHEN LENGTH(ESPANOLESHOMBRES) = 0 THEN 0 ELSE
ESPANOLESHOMBRES END AS INT) AS ESPANOLESHOMBRES,
     CAST(CASE WHEN LENGTH(ESPANOLESMUJERES) = 0 THEN 0 ELSE
ESPANOLESMUJERES END AS INT) AS ESPANOLESMUJERES,
    CAST(CASE WHEN LENGTH(EXTRANJEROSHOMBRES) = 0 THEN 0 ELSE
EXTRANJEROSHOMBRES END AS INT) AS EXTRANJEROSHOMBRES,
     CAST(CASE WHEN LENGTH(EXTRANJEROSMUJERES) = 0 THEN 0 ELSE
EXTRANJEROSMUJERES END AS INT) AS EXTRANJEROSMUJERES,
  FX_CARGA,
  FX_DATOS_INI,
  FX_DATOS_FIN
FROM padron_txt_2;
```

• 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.

2- Investigamos el formato columnar parquet.

• 2.1)
¿Qué es CTAS?

Es una operación que crea una nueva tabla a partir del resultado de una consulta 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 * 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 * FROM padron_txt_2;
```

• 2.4)
Opcionalmente también se pueden crear las tablas directamente desde 0 (en lugar de mediante CTAS) en formato parquet igual que lo hicimos para el formato txt incluyendo la sentencia STORED AS PARQUET. Es importante para comparaciones posteriores que la tabla padron_parquet conserve los espacios innecesarios y la tabla padron_parquet_2 no los tenga. Dejo a tu elección cómo hacerlo.

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

El formato columnar Parquet está diseñado para optimizar el rendimiento y la eficiencia en operaciones analíticas en entornos big data,
ofreciendo ventajas significativas en términos de almacenamiento, procesamiento y compatibilidad con diversos componentes del ecosistema big data.

• 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_txt: Tamaño total: 33,789,363 bytes (aproximadamente 32.2 MB)
padron_txt_2: Tamaño total: 24,225,909 bytes (aproximadamente 23.1 MB)
padron_parquet: Tamaño total: 2,520,393 bytes (aproximadamente 2.4 MB)
padron_parquet_2: Tamaño total: 2,508,363 bytes (aproximadamente 2.4 MB)

3- Juguemos con Impala.

• 3.1)
¿Qué es Impala?

Impala es parte del ecosistema de herramientas de Big Data de Apache y está diseñado para realizar consultas SQL interactivas y análisis en tiempo real de grandes conjuntos de datos.

• 3.2)
¿En qué se diferencia de Hive?

Impala se destaca por su rendimiento en consultas interactivas y acceso en tiempo real, mientras que Hive, aunque ha mejorado en términos de rendimiento,
sigue siendo más adecuado para tareas de procesamiento por lotes y análisis de datos a gran escala. La elección entre Impala y Hive dependerá de los requisitos específicos y del tipo de consultas que se planeen realizar.

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

Al ejecutar este comando en Impala, se descartará la información en caché relacionada con la tabla o tablas especificadas.
Cuando se realice una consulta subsiguiente que necesite metadatos, Impala volverá a cargar la información actualizada desde la fuente de datos subyacente.

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

```
INVALIDATE METADATA;
```

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

```
SELECT
  DESC_DISTRITO,
  DESC_BARRIO,
  SUM(ESPANOLESHOMBRES) AS Total_EspanolesHombres,
  SUM(ESPANOLESMUJERES) AS Total_EspanolesMujeres,
  SUM(EXTRANJEROSHOMBRES) AS Total_ExtranjerosHombres,
  SUM(EXTRANJEROSMUJERES) AS Total_ExtranjerosMujeres
FROM datos_padron.padron_parquet_2
GROUP BY DESC_DISTRITO, 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?

Principalmente la lentitud a la hora de enseñar la información de las tablas.

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

Al contrario que Hive, impala muestra los datos en segundos

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

Hay una gran diferencia a la hora de sacar las consultas

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 IF NOT EXISTS padron_particionado (
  COD_DISTRITO INT,
  COD_DIST_BARRIO INT,
  COD_BARRIO INT,
  COD_DIST_SECCION INT,
  COD_SECCION INT,
  COD_EDAD_INT INT,
  ESPANOLESHOMBRES INT,
  ESPANOLESMUJERES INT,
  EXTRANJEROSHOMBRES INT,
  EXTRANJEROSMUJERES INT,
  FX_CARGA TIMESTAMP,
  FX_DATOS_INI STRING,
  FX_DATOS_FIN STRING
)
PARTITIONED BY (DESC_DISTRITO STRING, DESC_BARRIO STRING)
STORED AS PARQUET;
```

• 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.mode=nonstrict;
SET hive.exec.max.dynamic.partitions=300;
SET hive.exec.max.dynamic.partitions.pernode=300;

INSERT OVERWRITE TABLE padron_particionado
PARTITION (DESC_DISTRITO, DESC_BARRIO)
SELECT
  COD_DISTRITO,
  COD_DIST_BARRIO,
  COD_BARRIO,
  COD_DIST_SECCION,
  COD_SECCION,
  COD_EDAD_INT,
  ESPANOLESHOMBRES,
  ESPANOLESMUJERES,
  EXTRANJEROSHOMBRES,
  EXTRANJEROSMUJERES,
  FX_CARGA,
  FX_DATOS_INI,
  FX_DATOS_FIN
FROM padron_parquet_2;
```

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

```
INVALIDATE METADATA;
```

• 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_DISTRITO,
  DESC_BARRIO,
  SUM(ESPANOLESHOMBRES) AS Total_EspanolesHombres,
  SUM(ESPANOLESMUJERES) AS Total_EspanolesMujeres,
  SUM(EXTRANJEROSHOMBRES) AS Total_ExtranjerosHombres,
  SUM(EXTRANJEROSMUJERES) AS Total_ExtranjerosMujeres
FROM
  padron_particionado
WHERE
    DESC_DISTRITO IN ('CENTRO', 'LATINA', 'CHAMARTIN', 'TETUAN', 'VICALVARO',
'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?

```
SELECT
  DESC_DISTRITO,
  DESC_BARRIO,
  SUM(ESPANOLESHOMBRES) AS Total_EspanolesHombres,
  SUM(ESPANOLESMUJERES) AS Total_EspanolesMujeres,
  SUM(EXTRANJEROSHOMBRES) AS Total_ExtranjerosHombres,
  SUM(EXTRANJEROSMUJERES) AS Total_ExtranjerosMujeres
FROM
  padron_particionado
WHERE
    DESC_DISTRITO IN ('CENTRO', 'LATINA', 'CHAMARTIN', 'TETUAN', 'VICALVARO',
'BARAJAS')
GROUP BY
  DESC_DISTRITO, DESC_BARRIO;
```

Esta primera consulta tarda alrededor de 46 segundos en sacar los datos.

```
SELECT
  DESC_DISTRITO,
  DESC_BARRIO,
  SUM(CAST(ESPANOLESHOMBRES AS INT)) AS Total_EspanolesHombres,
  SUM(CAST(ESPANOLESMUJERES AS INT)) AS Total_EspanolesMujeres,
  SUM(CAST(EXTRANJEROSHOMBRES AS INT)) AS Total_ExtranjerosHombres,
  SUM(CAST(EXTRANJEROSMUJERES AS INT)) AS Total_ExtranjerosMujeres
FROM
  padron_parquet
WHERE
    DESC_DISTRITO IN ('CENTRO', 'LATINA', 'CHAMARTIN', 'TETUAN', 'VICALVARO',
'BARAJAS')
GROUP BY
  DESC_DISTRITO, DESC_BARRIO;
```

Esta segunda no muestra ningún dato y aun así al trabajar por lotes tarda mucho en decirme que no hay datos.

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

Al igual que la anterior la segunda consulta no devuelve ningún dato y la primera de ellas es extremadamente rápida, ya que tarda
alrededor de 1 segundo en mostrar los datos.

• 4.7)
Hacer consultas de agregación (Max, Min, Avg, Count) tal cual el ejemplo anterior con las 3 tablas (padron_txt_2, padron_parquet_2 y padron_particionado) y comparar rendimientos tanto en Hive como en Impala y sacar conclusiones.

```
SELECT
  MAX(COD_EDAD_INT),
  MIN(COD_EDAD_INT),
  AVG(COD_EDAD_INT),
  COUNT(*)
FROM
  padron_txt_2;
```

```
SELECT
  MAX(COD_EDAD_INT),
  MIN(COD_EDAD_INT),
  AVG(COD_EDAD_INT),
  COUNT(*)
FROM
  padron_parquet_2;
```

```
SELECT
  MAX(COD_EDAD_INT),
  MIN(COD_EDAD_INT),
  AVG(COD_EDAD_INT),
  COUNT(*)
FROM
  padron_particionado;
```

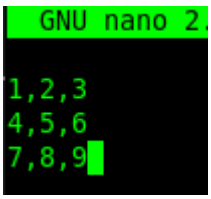
Al igual que en las otras consultas realizadas impala muestra los datos casi al instante mientras que Hive tarda mucho más.

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.

• 5.1)
Crear un documento de texto en el almacenamiento local que contenga una secuencia de números distribuidos en filas y separados por columnas, llámalo datos1 y que sea por ejemplo:
1,2,3
4,5,6
7,8,9

```
sudo nano ejercicios/datos1
```



• 5.2)
Crear un segundo documento (datos2) con otros números pero la misma estructura

```
sudo nano ejercicios/datos2
```

• 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.

```
hadoop fs -mkdir /user/cloudera/test
```

Comprobamos que se haya creado:

```
hadoop fs -ls /user/cloudera
```

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

```
hadoop fs -put /ejercicios/datos1 /user/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.

```
CREATE DATABASE IF NOT EXISTS numeros;

USE numeros;

CREATE TABLE numeros_tbl (
   columna1 INT,
   columna2 INT,
   columna3 INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\054'  -- \054 es el valor octal de la coma (,)
LINES TERMINATED BY '\n';
```

• 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?

```
LOAD DATA INPATH '/user/cloudera/test/datos1'
OVERWRITE INTO TABLE numeros_tbl;

DROP TABLE IF EXISTS numeros_tbl;
```

Si borramos la tabla, los datos en HDFS asociados a esa tabla también se borran.

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

```
hadoop fs -put /ejercicios/datos1 /user/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?

```
CREATE EXTERNAL TABLE datos1_tbl (
   columna1 INT,
   columna2 INT,
   columna3 INT
)

ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\054'  -- \054 es el valor octal de la coma (,)
LINES TERMINATED BY '\n';


DROP TABLE datos1_tbl;
```

Los datos no se borran de HDFS a menos que los borremos nosotros manualmente, para ver donde estan se puede utilizar el comando **DESCRIBE** acompañado del nombre de la tabla y nos mostrará su localización en HDFS.

• 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?

Borramos los datos

```
hadoop fs -rm /user/cloudera/test/datos1
```

Añadimos los datos de nuevo

```
hadoop fs -put /ejercicios/datos1 /user/cloudera/test/
```

Creamos la tabla externa:

```
CREATE EXTERNAL TABLE numeros_tbl (
   columna1 INT,
   columna2 INT,
   columna3 INT
)

ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\054'  -- \054 es el valor octal de la coma (,)
LINES TERMINATED BY '\n'
LOCATION '/user/cloudera/test/';
```

Hacemos el select:

```
SELECT * FROM numeros_tbl;
```

Podemos ver que la tabla tiene el contenido del archivo que hay en datos1.

• 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?

```
hadoop fs -put /ejercicios/datos2 /user/cloudera/test/
```

Al hacer el SELECT nuevamente se pueden ver los datos de los dos archivos en conjunto.

• 5.11)
Extrae conclusiones de todos estos anteriores apartados.

Las tablas gestionadas por Hive controlan la ubicación de los datos.
Las tablas externas pueden apuntar a ubicaciones específicas en HDFS.
Hive no elimina datos de HDFS cuando se borra una tabla externa.
Al cargar datos en una tabla gestionada, Hive los coloca en su propia ubicación en HDFS.
La ubicación de los datos en HDFS es crucial para Hive y cómo gestiona los datos.