Práctica Hive

1. **A partir de los datos (CSV) de Padrón de Madrid (https://datos.madrid.es/egob/catalogo/200076-1-padron.csv) llevar a cabo lo siguiente:**

    a. **Crear Base de datos datos_padron**

    ```
    DROP DATABASE datos_padron CASCADE;
    CREATE DATABASE IF NOT EXISTS datos_padron;
    USE datos_padron;
    ```

    b. **Crear tabla 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 carácter ';' y los campos estarán encerrados en comillas dobles '"' y se deberá omitir la cabecera del fichero de datos al crear la tabla.**

    ```
    -- Creamos la tabla
    DROP TABLE IF EXISTS datos_padron.padron_txt;
    CREATE TABLE IF NOT EXISTS datos_padron.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' -- Utilizamos
    SerDe para posteriormente cargar el archivo CSV
    WITH SERDEPROPERTIES
    (-- Propiedades para que delimite cada campo entre ';' , lea cada campo encerrado
    entre ""  , y lea el salto de linea
    "separatorChar" = ";",
    "quoteChar" = "\"",
    "escapeChar" = "\n"
    )
    STORED AS TEXTFILE --Tabla tipo txt
    TBLPROPERTIES("skip.header.line.count"="1",      -- Propieda para tratar los campos
    con espacio como null
            "seriealization.null.format" = ""); -- Propiedad para omitir encabezado
    ```

```
-- Cargar los datos desde local (Ejecutar en la shell con usuario root, en HUE no
funciona)
LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Datasets HIVE/data.csv'
OVERWRITE INTO TABLE datos_padron.padron_txt;
--verificamos
select * from padron_txt limit 5;
```

Otra forma de cargar los datos directamente sería crearla como tabla externa y apuntar en location hacia el directiorio donde está el archivo CSV

c. **Crear tabla padron_txt2 que haga trim sobre los datos de algunas columnas con la finalidad de eliminar los espacios en blanco innecesarios. (A través de CTAS -> crea una tabla a partir de la estructura de un select).**

```
-- Creamos la tabla elminando los espacios en blanco con trim, no es necesario el
SerDe puesto que hacemos un select de la tabla anterior con los datos csv cargados
DROP TABLE IF EXISTS padron_txt2
CREATE TABLE padron_txt2
 AS
SELECT
  COD_DISTRITO,
  trim(DESC_DISTRITO) DESC_DISTRITO,  -- trim(col) elimina espacios
  COD_DIST_BARRIO,
  trim(DESC_BARRIO) DESC_BARRIO,     -- trim(col) elimina espacios
  COD_BARRIO,
  COD_DIST_SECCION,
  COD_SECCION,
  COD_EDAD_INT,
  EspanolesHombres,
  EspanolesMujeres,
  ExtranjerosHombres,
  ExtranjerosMujeres
FROM padron_txt;
-- verificamos
select * from padron_txt2 limit 5;

-- TENEMOS QUE CONVERTIR LAS COLUMNAS NECESARIAS A INT POR QUE EL SERDE
LAS PASA TODAS A STRING. Creamos tabla padron_txt3 casteando a mano todas las
columnas
-- Supongo que en versiones superiores de hive o con otros softwares como spark no
sería necesario
DROP TABLE IF EXISTS padron_txt3
CREATE TABLE padron_txt3
 STORED AS TEXTFILE --Tabla tipo txt
 AS
SELECT
  CAST(COD_DISTRITO AS INT) COD_DISTRITO,
```

```
 trim(DESC_DISTRITO) DESC_DISTRITO,
 CAST(COD_DIST_BARRIO AS INT) COD_DIST_BARRIO ,
 trim(DESC_BARRIO) DESC_BARRIO,
 CAST(COD_BARRIO AS INT) COD_BARRIO,
 CAST(COD_DIST_SECCION AS INT) COD_DIST_SECCION,
 CAST(COD_SECCION AS INT) COD_SECCION,
 CAST(COD_EDAD_INT AS INT) COD_EDAD_INT,
 CAST(EspanolesHombres AS INT) EspanolesHombres,
 CAST(EspanolesMujeres AS INT) EspanolesMujeres,
 CAST(ExtranjerosHombres AS INT) ExtranjerosHombres,
 CAST(ExtranjerosMujeres AS INT) ExtranjerosMujeres
FROM padron_txt2; -- apuntamos a txt2 para coger la tabla con espacios ya
eliminados
```

d. Investigar y entender la diferencia de incluir la palabra LOCAL en el comando LOAD DATA

La palabra LOCAL DENTRO DE LOAD DATA INPATH indica que la carga de datos/archivo se hará desde el sistema de archivos local, si no se especifica, la carga de datos/archivo se hará desde HDFS.

2. ¿Qué es CTAS?

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

```
-- Introducimos el formato columnar parquet en STORED AS. La hacemos
referenciando a padron_txt3 que no tiene espacios y tiene bien delimitadas las
columnas
DROP TABLE IF EXISTS  padron_parquet
CREATE TABLE padron_parquet
 STORED AS PARQUET --Tabla tipo parquet
 AS
SELECT *
FROM padron_txt3; --apuntamos a txt3 para coger los datos limpios (sin espacios y
columnas bien delimitadas
```

b. Crear Tabla (en Hive) padron_parquet2 (cuyos datos serán almacenados en el formato columnar parquet) a partir de la tabla padron_txt2 mediante un CTAS.

```
-- Realmente no necesario porque con padron_parquet ya cargamos los datos
limpios
DROP TABLE IF EXISTS padron_parquet2
CREATE TABLE padron_parquet2
 STORED AS PARQUET --Tabla tipo parquet
 AS
SELECT *
```

FROM padron_txt3; --utilizo parquet_txt3 porque tiene las columnas int bien definidas

c. **Investigar en que consiste el formato columnar parquet y las ventajas de trabajar con este tipo de formatos.**

Parquet es un formato de almacenamiento basado en columnas para Hadoop mientras que **Avro** es un formato basado en filas. Parquet es más utilizado cuando los datos tienen muchas columnas, su uso esta generalizado a trabajar con un subconjunto de estas columnas en lugar de todos los registros completos.

d. **Comparar el tamaño de los ficheros de los datos de las tablas padron_txt (CSV) y padron_parquet (alojados en hdfs cuya ruta se puede obtener de la propiedad location de cada tabla por ejemplo haciendo SHOW CREATE TABLE).**

Los ficheros almacenados en HDFS de la tabla padron_txt ocupan 22mb porque es en el que cargamos desde local el archivo csv, los que hacen referencia a la tabla padron_txt2 (en la que eliminamos los espacios en blanco con la función trim) ocupan 11.6mb y los que hacen referencia a la tabla padron_parquet ocupan solo 1mb. El formato de columnas parquet tiene mucha más eficiencia a la hora de comprimir.

e. **Comparar el tamaño de los ficheros de los datos de las tablas padron_txt (CSV), padron_txt2, padron_parquet y padron_parquet2 (alojados en hdfs cuya ruta se puede obtener de la propiedad location de cada tabla por ejemplo haciendo SHOW CREATE TABLE).**

- Padron_txt 22mb
- Padron_txt2 11.6mb
- Padron_parquet 931.8kb
- Padron_parquet2 929.7kb

Utilizamos el visor de archivos de HDFS de la izquierda. Como hemos comentado anteriormente, parquet optimiza mucho más la comprensión y por ello hace que ocupe alrededor de 1mb. La diferencia entre padron_parquet (931.8kb) y paddron_parquet2 (929.7kb) estaría en que la segunda hace refencia a la tabla en la que se han eliminado los espacios en blanco con la función trim (padron_txt2).

3. Impala

a. ¿Qué es impala?

Es un motor de consultas de bbdd diseñado para optimizar la latencia de las consultas SQL en Hadoop, es una alternativa a Hive para consultas menos pesadas.

Impala ejecuta las consultas directamente en el cluster en lugar de ejecutar un MapReduce para procesar**, cargando los datos en memoria y permaneciendo allí toda la fase de consulta**, lo que le hace ser más rápido y efectivo que Hive.

Impala requiere que la mayor parte de los datos quepan en la memoria principal, servidores con más capacidad de memoria que los de hadoop en mapreduce. Si un nodo cae, la consulta no puede lanzarse y lanza un error. Se recomienda para consultas rápidas y que se puedan reiniciar.

b. ¿En qué se diferencia a Hive?

Hive e Impala se suelen utilizar juntas en proyectos. Las principales diferencias radican en Impala no soporta algunas funcionalidades que Hive sí, aunque poco a poco las van implementando. Algunas de ellas pueden ser:

- El tipo de datos date
- Funciones XML y JSON, algunas funciones de agregación, UDFs
- Sampling (ejecutar querys sobre una muestra de la tabla)

c. Comando INVALIDATE METADATA, ¿en qué consiste?

Marca los metadatos de una o todas las tablas como obsoletos, por lo que la próxima vez que Impala realiza una consulta en una tabla con este comando aplicado, vuelve a cargar los metadatos antes de continuar con la consulta.

Se trata de una operación muy costosa en comparación con la actualización incremental de datos con REFRESH, por lo que tendremos que elegir entre estas dos para actualizar los metadatos.

d. Hacer invalidate metadata en Impala de Base de datos datos_padron

INVALIDATE METADATA

e. Calcular el total de EspanolesHombres, EspanolesMujeres, ExtranjerosHombres y ExtranjerosMujeres agrupado por DESC_DISTRITO y DESC_BARRIO.

i. Llevar a cabo la consulta en **Hive** en las tablas padron_txt y padron_parquet. ¿Alguna conclusión?

```
-- Select de tabla parquet o txt (cambiar nombre en FROM)
SELECT DESC_DISTRITO,
    DESC_BARRIO,
    SUM(espanolesHombres) Tot_Esp_H,
    SUM(espanolesMujeres) Tot_Esp_M,
    SUM(extranjerosHombres) Tot_Ext_H,
    SUM(extranjerosMujeres) Tot_Ext_M
FROM padron_txt
GROUP BY DESC_DISTRITO, DESC_BARRIO;
```

Query padron_txt -> Time taken: **34.971 seconds**, Fetched: 133 row(s)
Query padron_parquet -> Time taken: **22.654 seconds**, Fetched: 133 row(s)

Se observa que para el formato de tabla columnar parquet HIVE es más rapido procesando la query. (35segs en padron_txt y 23 segs en padron_parquet)

ii. **Llevar a cabo la consulta en Impala en las tablas padron_txt y padron_parquet. ¿Alguna conclusión?**

Impala no soporta el formato SerDe aplicado en la tabla padron_txt por lo que esta consulta solo se podría hacer en Hive.

Por otro lado, la tabla padron_parquet esta creada con CTAs a través de la tabla padron_txt. Esto implica que el SerDe al ser aplicado ha transformado todas las columnas a tipo STRING, por lo que para poder lanzar esta query en impala habría que pasar dichas columnas a tipo INT **(creadas ya en tablas padron_txt3 y padron_parquet2).**

Si realizamos las dos consultas se nota una mejora de tiempo de ejecución en la query de impala sobre hive

iii. **¿Se percibe alguna diferencia de rendimiento entre Hive e Impala?**

Impala aun **no soporta algunas funcionalidades que Hive sí**, pero es más **rápido procesando querys**, por lo que es recomendable usar Hive salvo que sepas que esa query está bien optimizada en Impala.

iv. **Obtener alguna métrica más adicional que nos sean de interés (Sitios con pocos empadronados, que distrito o barrio con más extranjeros españoles, etc.**

```
-- Valores medios en Extranjeros y Españoles agrupado por distrito y barrio
SELECT DESC_DISTRITO,
    DESC_BARRIO,
    AVG(espanolesHombres) EspanolesHombres,
    AVG(espanolesMujeres) EspanolesMujeres,
    AVG(extranjerosHombres) ExtranjerosHombres,
    AVG(extranjerosMujeres) ExtranjerosMujeres
FROM padron_parquet2
GROUP BY DESC_DISTRITO, DESC_BARRIO;
```

4. **Particionamiento**

   a. **Crear tabla (Hive) padron_particionado particionada por los campos DESC_DISTRITO y DESC_BARRIO cuyos datos estén en formato parquet.**

```
-- Creamos tabla padron_particionado y padron_particionado_txt (cambiar formato y nombre)
DROP TABLE IF EXISTS padron_particionado_txt;

CREATE TABLE padron_particionado_txt --Cambiamos nombre (txt o parquet)
(
 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
)
PARTITIONED BY ( DESC_DISTRITO STRING,
                 DESC_BARRIO STRING )
STORED AS TEXTFILE; -- Cambiamos formato (PARQUET O TEXTFILE)
```

   b. **Insertar datos (en cada partición) dinámicamente (con Hive) en la tabla recién creada a partir de un select de la tabla padron_parquet2.**

```
-- seteamos estas propiedades que permite realizar el particionado dinamico
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions = 10000;
set hive.exec.max.dynamic.partitions.pernode = 1000;

-- seteamos mas meroia ram (por defecto la configuracion de hadoop trae 1gb)
set mapreduce.map.memory.mb = 2048;
set mapreduce.reduce.memory.mb = 2048;
set mapreduce.map.java.opts=-Xmx1800m

-- Insertamos a traves de los datos de la tabla padron_txt (formato txt)
--Importante el orden de las columnas
INSERT OVERWRITE TABLE padron_particionado_txt -- cambiar nombre tabla
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,
 DESC_DISTRITO,
 DESC_BARRIO
FROM padron_txt3; -- cambiar tabla a txt o parquet

 -- Insertamos a traves de los datos de la tabla padron_parquet2 (formato parquet)
INSERT OVERWRITE TABLE padron_particionado_parquet
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,
 DESC_DISTRITO,
 DESC_BARRIO
FROM padron_parquet;

-- verificamos las particiones (las vemos en el browser tambien en wharehouse)
SHOW PARTITIONS padron_particionado_parquet;
SHOW PARTITIONS padron_particionado_txt;
```

c. **Hacer invalidate metadata en Impala de Base de datos padron_particionado.**

```
--HACER EN IMPALA
INVALIDATE METADATA;
```

d. 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.

i. Llevar a cabo la consulta en Hive en las tablas padron_parquet y padron_particionado. ¿Alguna conclusión?

```
-- Para la tabla padron_parquet (28 segundos)
-- Para la tabla padron_parquet_particionado (15 segundos)
```

-- Como podemos observar las particiones por distrinto y barrio hacen que sean más rapidas las consultas.

```
SELECT
 DESC_DISTRITO,
 DESC_BARRIO,
 SUM(EspanolesHombres) TotalEsp_H,
 SUM(EspanolesMujeres) TotalEsp_M,
 SUM(ExtranjerosHombres) TotalExt_H,
 SUM(ExtranjerosMujeres) TotalExt_M
FROM padron_particionado_parquet -- cambiar (padron_parquet o padron_particionado_parquet)
-- En lugar de usar = o LIKE y repetir la vairble cogemos IN y ponemos los valores en una lista
WHERE DESC_DISTRITO IN ('CENTRO','LATINA','CHAMARTIN','TETUAN','VICALVARO','BARAJAS')
GROUP BY DESC_DISTRITO, DESC_BARRIO;
```

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

-- Para la tabla padron_parquet (28 segundos)
-- Para la tabla padron_parquet_particionado (15 segundos)

-- Como podemos observar las tablas con particiones son mas rápidas querys en Impala son mas rápidas.

```
SELECT
 DESC_DISTRITO,
 DESC_BARRIO,
 SUM(EspanolesHombres) TotalEsp_H,
 SUM(EspanolesMujeres) TotalEsp_M,
 SUM(ExtranjerosHombres) TotalExt_H,
 SUM(ExtranjerosMujeres) TotalExt_M
FROM padron_particionado_parquet -- cambiar (padron_parquet o padron_particionado_parquet)
-- En lugar de usar = o LIKE y repetir la vairble cogemos IN y ponemos los valores en una lista
WHERE DESC_DISTRITO IN ('CENTRO','LATINA','CHAMARTIN','TETUAN','VICALVARO','BARAJAS')
GROUP BY DESC_DISTRITO, DESC_BARRIO;
```

e. Hacer consultas de agregación (Max, Min, Avg, Count) tal cual el ejemplo anterior con las 3 tablas (padron_txt, padron_parquet y padron_particionado) y comparar rendimientos tanto en Hive como en impala y sacar conclusiones.

-- Agrupamos en distritos y barrios con más extranjeros (Hombre, mujer y total) en orden ascendente

```
SELECT DESC_DISTRINTO,
        DESC_BARRIO,
        Max(ExtranjerosHombres) Ext_H
        Max(Extranjeros Mujeres) Ext_M
        Ext_H + Ext_M AS Tot_Ext
FROM  padron_particionado_parquet – podemos cambiar por padron_txt, padron_parquet y padron_particionado_parquet
GROUP BY DESC_DISTRINTO, DESC_BARRIO
SORTED BY
```

```
-- Distritos y Barrios agrupados por Población total de empadronados ordenados de menor a mayor
SELECT  DESC_DISTRITO, DESC_BARRIO,
     (SUM(ExtranjerosHombres) + SUM(ExtranjerosMujeres) + SUM(EspanolesHombres) + SUM(EspanolesMujeres)) AS 'Poblacion'

FROM padron_parquet2
GROUP BY DESC_DISTRITO, DESC_BARRIO
ORDER BY Poblacion ASC
```

```
-- Distritos/Barrios del total de extranjeros ordenados de mayor a menor
SELECT  DESC_DISTRITO, DESC_BARRIO,
     (SUM(ExtranjerosHombres) + SUM(ExtranjerosMujeres)) AS Tot_Ext,
     SUM(ExtranjerosHombres) AS Extranjeros_Hombres,
     SUM(ExtranjerosMujeres) AS Extranjeros_Mujeres
FROM padron_parquet2
GROUP BY DESC_DISTRITO, DESC_BARRIO
ORDER BY Tot_Ext DESC
```

```
--Población Media agrupada por distrito y barrio
SELECT  DESC_DISTRITO, DESC_BARRIO,
     AVG(ExtranjerosHombres) Avg_Ext_H,
     AVG(ExtranjerosMujeres) Avg_Ext_M,
     AVG(EspanolesHombres) Avg_Esp_H,
     AVG(EspanolesMujeres) Avg_Esp_M
FROM padron_parquet2
GROUP BY DESC_DISTRITO, DESC_BARRI
```

En cuanto a las clonclusiones, comparando con esta query observamos:

- Mucha diferencia en tiempos de ejecución de querys entre Hive /Impala (siendo Impala mucho más rápido, pero soportando menos funcionalidades (función SORT BY por ejemplo).

- En cuanto a formatos columnares de tabla (txt, parquet) se aprecian mas rapidez en tiempos de ejecucion de querys en parquet.

- En cuanto a particionado/no particionado, se aprecia más rapidez en tiempos de ejecución de querys en las tablas particionadas.

- Finalmente creo que lo más óptimo para trabajar con Hive/Impala es utilizar las consultas más rápidas y que se puedan volver a lanzar sin problema en Impala, usando el formato de columna parquet y el particionamiento para mejorar el rendimiento.

Información:

- https://cwiki.apache.org/confluence/collector/pages.action?key=Hive
- https://cwiki.apache.org/confluence/display/Hive/GettingStarted#GettingStarted-CreatingHiveTables
- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL
- https://cwiki.apache.org/confluence/display/Hive/DynamicPartitions