Formato Columnar de Ficheros

¿Cómo procedemos en un análisis de datos?

- El primer paso es dialogar con los datos: entender qué datos hemos recibido y qué información aportan.
- A continuación procedemos a revisar y filtrarlos siguiendo un posible control de calidad, o formato, etc.
- Ajustamos el juego de datos a la hipótesis que queremos comprobar, al proyecto, etc.: lo adaptamos a su principal funcionalidad en el entorno correspondiente
- Una vez los hemos preparado nos ponemos a trabajar con ellos: análisis estadísticos, aplicando modelos, aplicando técnicas de visualización, etc.

Formato de Ficheros

- Al final la información queda almacenada utilizando registros que son acumulados uno tras otro en formato secuencial
- La lectura queda entonces condicionada de la misma manera: se procede a leer los registros uno tras otro y seleccionamos la información que nos interesa (y no utilizamos mucha información que ha tenido que ser leída obligatoriamente)
- Sin embargo, en los cálculos habituales en entornos analíticos suele interesar coger un valor o un dato específicamente de cada registro con lo que el proceso de lectura es muy ineficiente (se lee mucha información que no nos interesa)
- Si representamos la información como si fuera una tabla podemos decir que leemos todos los registros (filas) para solo seleccionar algunos parámetros (columnas) de cada uno de todos los registros

Formato orientado a registro

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

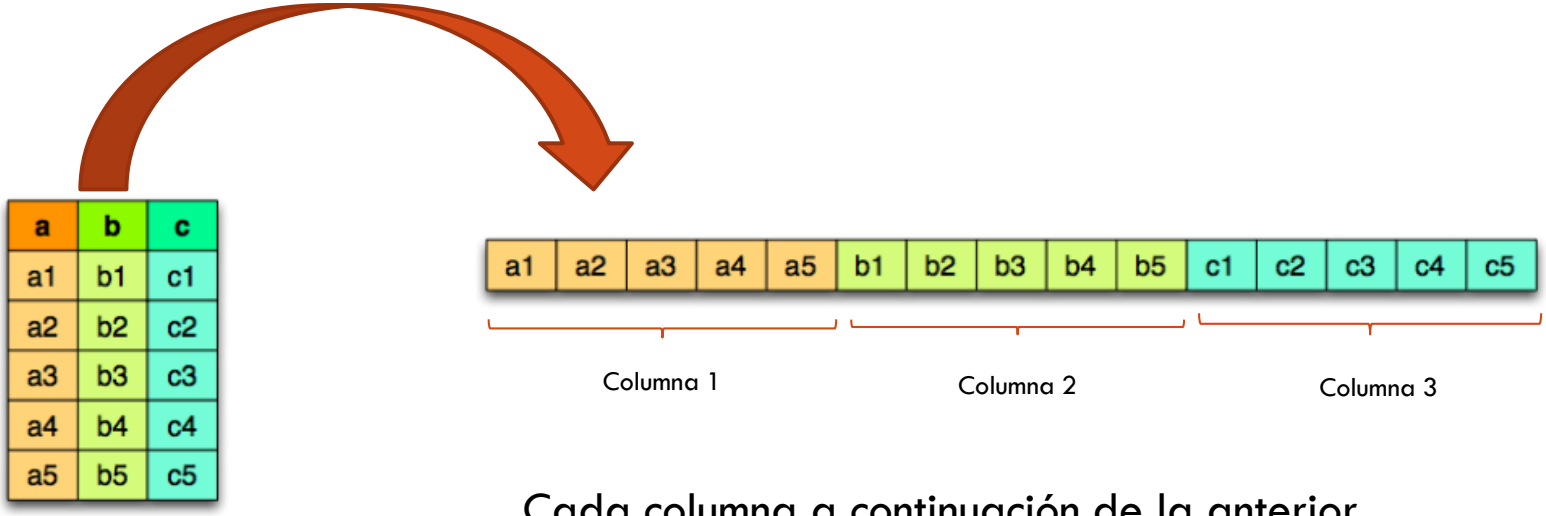
Representación lógica de una tabla

Cada registro a continuación del anterior

¿Qué supone seleccionar una determinada columna?

¿Y añadir nuevos registros?

Formato Columnar



Representación lógica de una tabla

Cada columna a continuación de la anterior

¿Qué supone seleccionar una determinada columna?

¿Y añadir nuevos registros?

Formato útil tanto en disco como en memoria

En disco. Permanente.

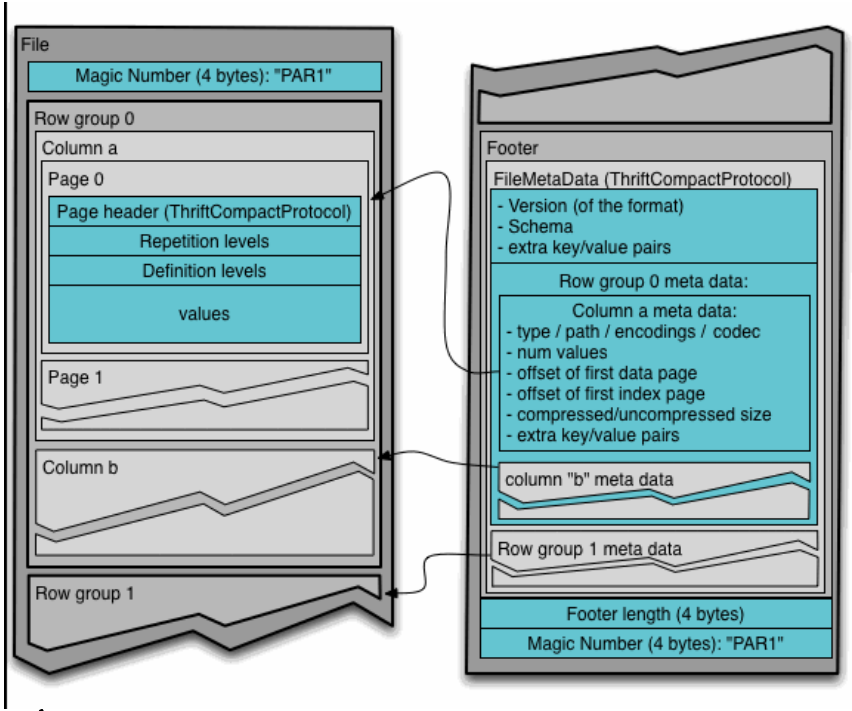
- Accedido por múltiples queries.
- Es prioritario reducir I/O (aunque seguimos necesitando una buena CPU).
- Principalmente acceso de tipo continuo.

En memoria: Transitorio.

- Formato compartido por muchas implementaciones
- Prioritario una buena CPU (aunque sigues necesitando un buen I/O).
- Acceso en continuo pero también aleatorio.

Formato Parquet

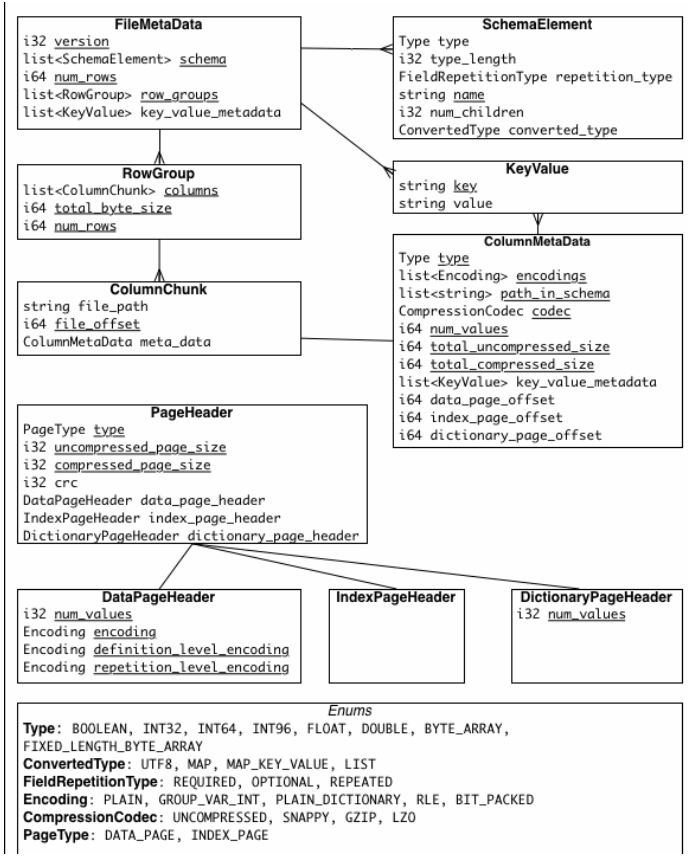
- Estructuras anidadas de datos
- Formato compacto:
  - Codificación dependiendo del tipo
  - Permite compresión
- I/O optimizado:
  - Projection push down (column pruning)
  - Predicate push down (filtros basados en estadísticas)



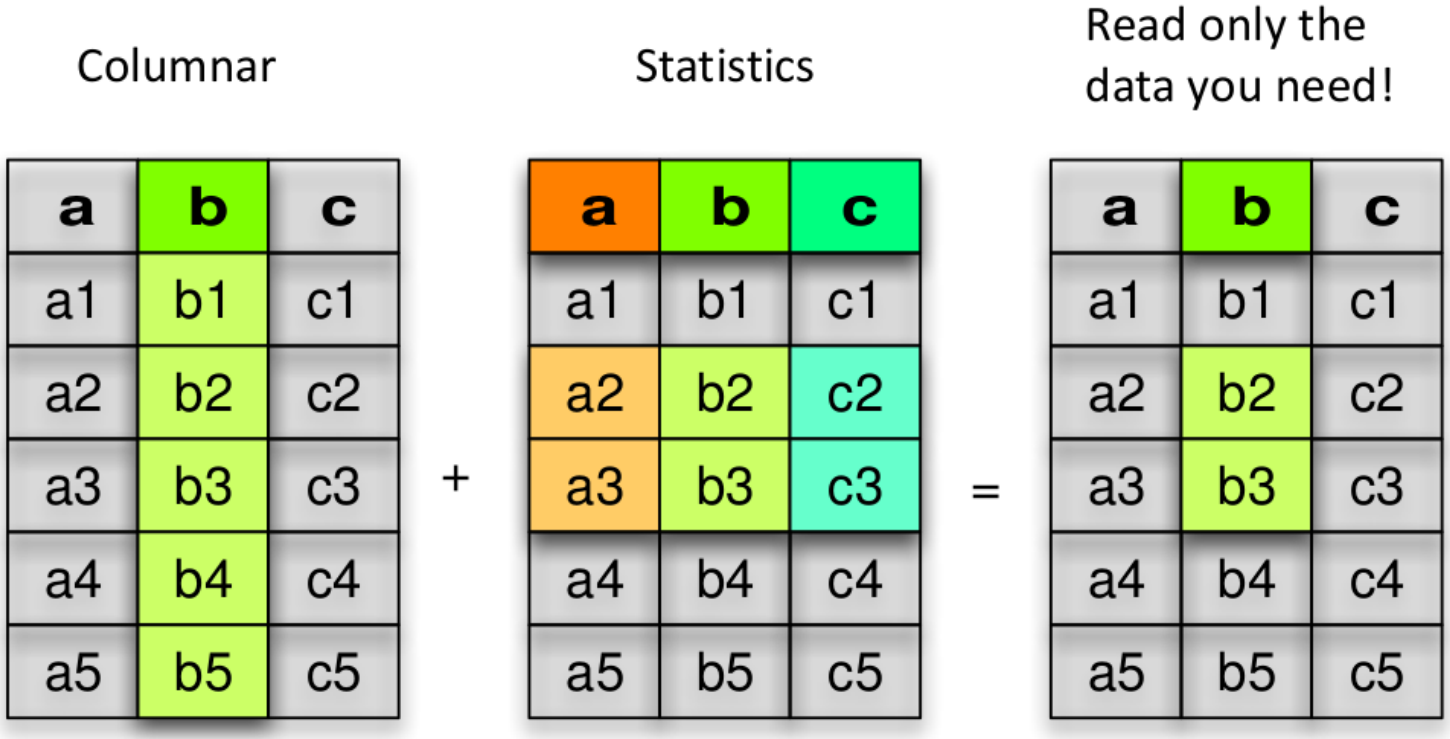
Metadatos

Hay tres tipos de metadatos:

- Fichero
- Columna (chunk)
- Cabecera de página/page header (los chunks de columnas están compuestas por páginas consecutivas). Las páginas tienen una cabecera común que permite identificar en el momento de lectura si nos interesa o no



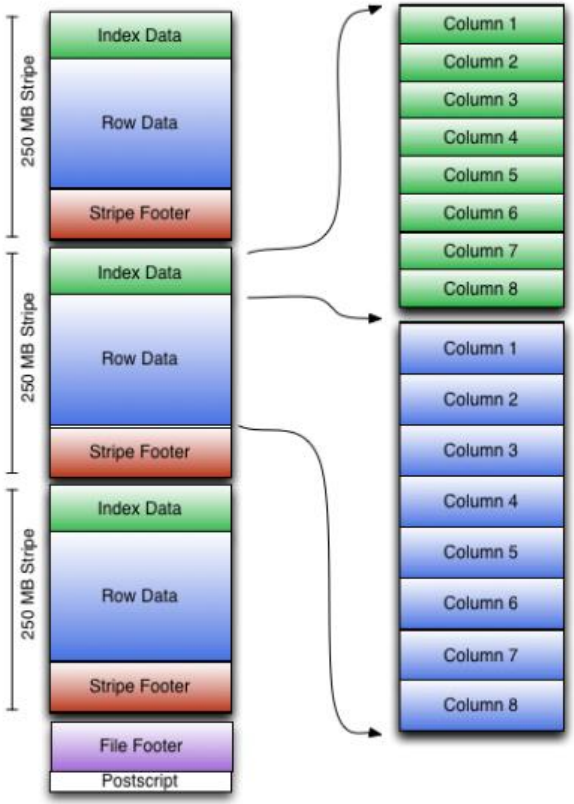
Accedemos solo a los datos necesarios



Optimized Row Columnar (ORC) Format

- Columnas almacenadas por separado
- Tipos de datos habituales
  - Utiliza codificadores específicos para el tipo de dato
  - Almacena tambien estadísticas (min, max, sum, count)
- Tiene tambien un índice sencillo
  - Que le permite saltar bloques de filas que no interesan
- Utiliza bloques lógicos de mayor tamaño que los de HDFS
  - 256 MB por defecto
  - Tiene índices para manejar los límites de los bloques

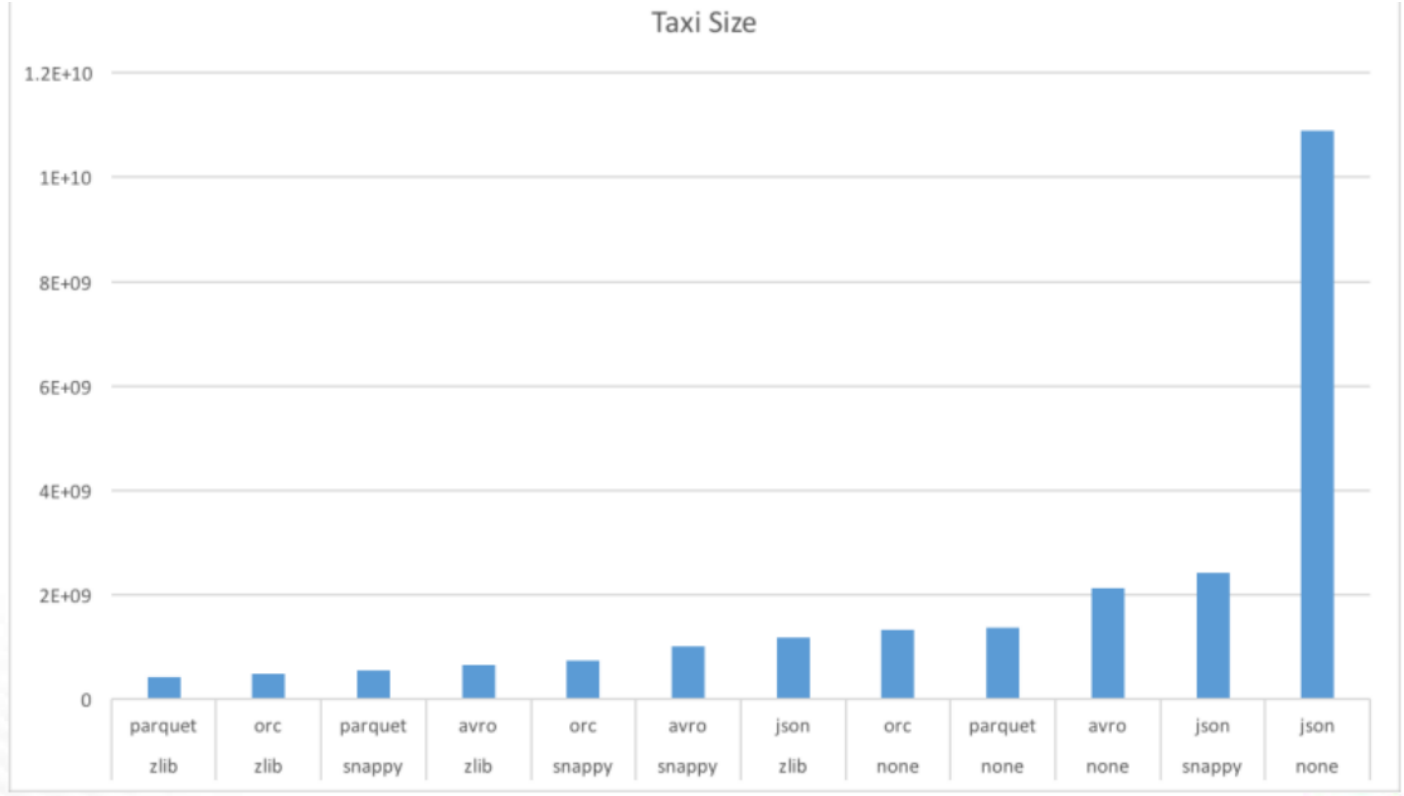
File Layout


250 MB Stripe
Index Data
Row Data
Stripe Footer
250 MB Stripe
Index Data
Row Data
Stripe Footer
250 MB Stripe
Index Data
Row Data
Stripe Footer
File Footer
Postscript
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8

Compresión

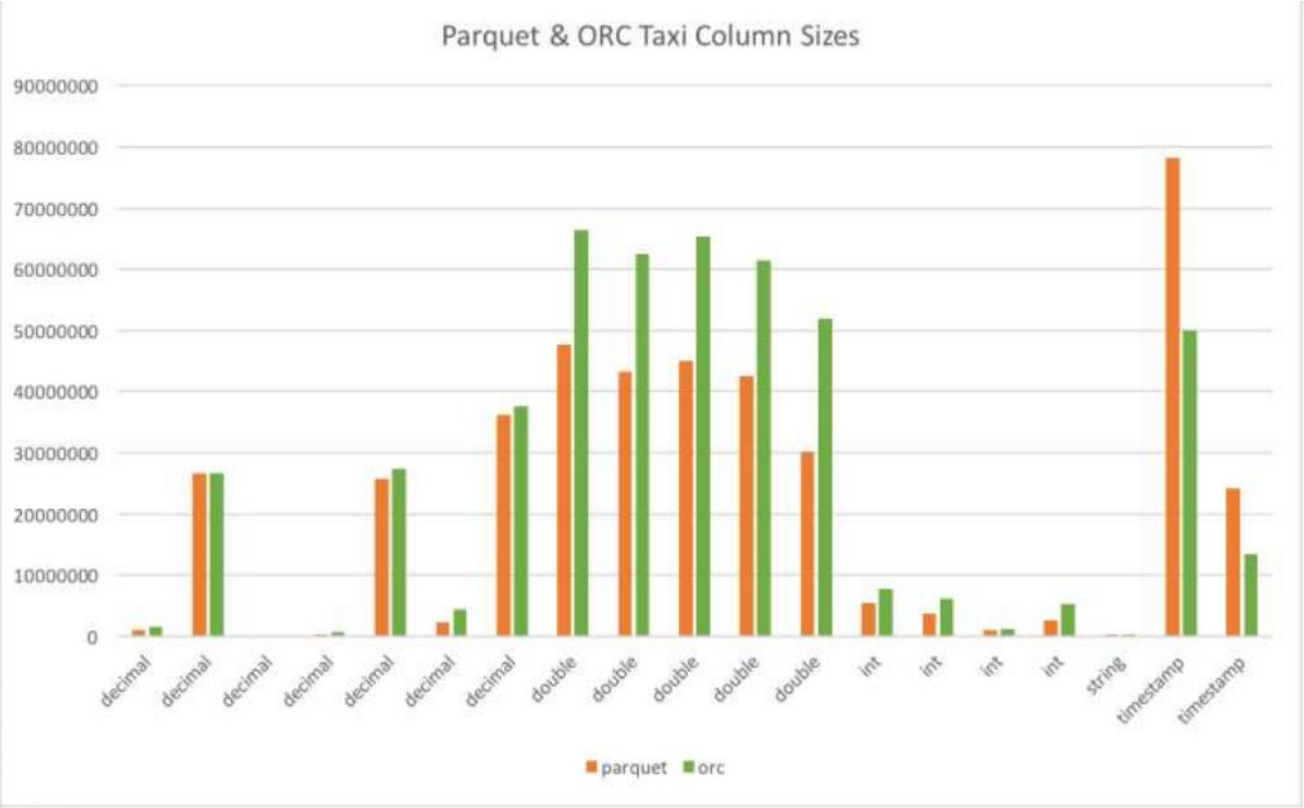
- El tamaño del dato sí importa
  - Hadoop almacena todos los datos pero requiere hardware
  - Importante el factor de velocidad de lectura
- ORC y Parquet utilizan RLE (Run-Length Encoding) y Diccionarios
- Todos los formatos tienen la opción de comprimir los datos
  - ZLIB (GZip) – compresión más fuerte y proceso, entonces, más lento
  - Snappy – no tan fuerte y más rápido

Benchmark de nivel de compresión



https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Comparando Parquet y ORC



¿Cómo podemos utilizar formato columnar?

- Tanto Python, como R y Spark tienen su implementación del formato columnar adaptadas a su manejo de estructuras de datos (tanto en persistencia en disco como en memoria)
  - Python: Fastparquet, PyArrow

- Pero también se puede utilizar en otros entornos de persistencia especiales como Hadoop.

Opciones de líneas de comando

- Existe la opción de “manejar” ficheros parquet desde línea de comandos, por ejemplo:
  - parquet-tools que están disponibles en java
    - Repositorio en git-hub (parquet-mr)
    - Disponible el código fuente a compilar con Maven (también se necesita el compilador de thrift)
  - parquet-cli
    - En python: pip install parquet-cli (require pandas)
    - Linea de commando (parq) basado en python para poder leer ficheros parquet

Probemos alguna de esta funcionalidad

- Utilizaremos Python y Jupyter Notebooks (Anaconda)
- Instalamos Fastparquet y Pyarrow
- Utilicemos dos juegos de datos:
  - Un fichero de clientes que utilizaremos en otros entornos
  - Un fichero de Movielens (kaggle) sobre el que podremos hacer algunas pruebas (bajadlo de https://grouplens.org/datasets/movielens/)

- En Moodle tenéis los notebooks

Qué veremos

1. Cómo se puede crear un fichero parquet desde Python con la librería pandas

2. Revisar diferencias entre SQL y Pandas a la hora de manejar los datos

Qué hemos podido comprobar

- Manejo de DataFrames versus SQL
- Tiempos manejando fichero ratings de Movielens (26 Millones de registros, 700 MB)
  - Pandas, Fastparquet
    - De creación de DF a partir de CSVs — 10.96 Segs.
    - De creación de un fichero parquet a partir de un DF — 26.88 Segs.
    - De creación de DF a partir de un fichero parquet — **1.21 Segs.**
  - Pyarrow
    - De creación de DF a partir de CSVs — **1.40 Segs.**
    - De creación de DF a partir de parquet — **1.10 Segs.**

Repaso. HiveQL

- Utiliza los conceptos de BBDD relacionales: Tablas, columnas, vistas, etc.
- Diseño para manejar datos estructurados.
- Tiene algunas variaciones respecto al SQL de una BBDD relacional.
- Traduce las sentencias SQL en programas ejecutables en YARN, Spark…

```
CREATE TABLE movies (movie_id int, movie_title string, release_date  string, video_release_date  string, IMDB_URL
string, unknown int, Action int, Adventure int, Animation int, Childrens int, Comedy int, Crime int, Documentary int,
Drama int, Fantasy int, FilmNoir int, Horror int, Musical int, Mystery int, Romance int, SciFi int, Thriller int, War
int, Western int)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '|'
  STORED AS TEXTFILE;

LOAD DATA INPATH 'hdfs:///user/vagrant/movielens/ml-100k/u.item' OVERWRITE INTO TABLE movies;

SELECT * FROM movies WHERE Drama = 1;
```

Record Format

- Los valores se escriben y leen en los ficheros de las tablas utilizando un SerDe
  - Serializador/Deserializador – para codificar y decodificar los valores

- El Deserializador coge una cadena o una representación binaria de un registro y la traduce en un objeto Java que Hive pueda manipular. Generalmente está involucrado en consultas para ejecutar las sentencias de los *Select*
- El Serializador, por contra, coge el objeto Java de Hive y lo convierte en algo que Hive pueda escribir en HDFS o cualquier otro Sistema (o protocolo). Habitualmente se utilizan cuando escribimos datos como por ejemplo instrucciones Insert-Select.

Record Format

- Otras implementaciones de SerDe
  - Parquet
  - ORC (Optimized Row Columnar)
  - CSV
  - Thrift (ThriftSerDe) – Thrift binary encoding
  - Regular Expressions (RegexSerDe) – Usar regex para extraer campos
  - Hive Binary format (LazyBinarySerDe) – Binary storage
- Se pueden desarrollar SerDe específicos para cada situación
- Otros disponibles en Internet
  - JSON
  - Avro
  - Etc.

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-RowFormats&SerDe

¿Y cómo hacemos para que convivan Hive y Parquet?

- Opción 1
  - Generamos un fichero parquet en cualquier plataforma
  - Lo “subimos” a hdfs
  - Definimos la tabla en HIVE como, por ejemplo, un external table.
- Opción 2
  - Tenemos una tabla ya creada en Hive (en formato no-parquet)
  - Creamos una tabla en formato parquet y la rellenamos desde la no-parquet. Utilizamos un procedimiento denominado CTAS (Create Table as Select)

Ejemplo con Parquet

```
create table ratings (
            userId int,
            movieId int,
            rating double,
            ratedate bigint )
row format delimited
fields terminated by ','
stored as textfile
tblproperties("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH '/tmp/ratings.csv' OVERWRITE INTO TABLE ratings;

CREATE TABLE ratings_parquet STORED AS PARQUET AS SELECT * FROM ratings;
```

Los SERDE pueden ayudar en otros casos: HIVE y Json

- Se utiliza un SerDe específico para JSON
- El SerDe se encarga de interpretar el registro de entrada como que está en formato Json y su salida

Ejemplo con Json

```
CREATE TABLE jsontable (
        campo_1 boolean,
        campo_2 array<string>,
        campo_3 double,
        campo_4 string )
      ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
      STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/tmp/Test.json' into Table jsontable;
```

Practicando estos conceptos (I/IV)

1. Uso de SerDe para almacenar información en formato parquet
   - Creándolo como CTAS
   - Creando una tabla externa

2. Uso de SerDe para almacenar en Hive registros JSON

Practicando estos conceptos (II/IV)

- Descarga los ficheros zip de Moodle con los ficheros de la práctica
- Cópialos en la imagen virtual (es conveniente compartir un directorio con la imagen virtual para hacerlo todo más fácil)
- Arranca una terminal en la imagen virtual
- Confirma que los contenedores están arrancados

```
docker stats
docker-compose top
```

Practicando estos conceptos (III/IV)

- Abre una Shell en el Docker de Hive:

```
docker-compose exec hive-server bash
```

- Invoca el intérprete de Hive (ya sea hive o beeline) con comandos del tipo:

```
/opt/hive/bin/beeline --color=true -u jdbc:hive2://localhost:10000 -f fichero.hql
/opt/hive/bin/beeline --color=true -u jdbc:hive2://localhost:10000 -e 'Sentencia SQL;'
```

Características de tablas en Hive (IV/IV)

- Dentro de la Shell de hive podemos utilizar estos comandos para comprobar la definición de la tabla:

```
describe nombre_de_tabla;
describe formatted nombre_de_tabla;
describe extended nombre_de_tabla;
```