Introducción

Si alguna vez has empleado la función **pd.read_csv()** de la **librería pandas** habrás notado que cuando el archivo CSV tiene muchas filas la lectura de la información es muy lenta.

La realidad es que esta función puede ejecutarse aproximadamente 50 veces más rápido si utilizamos el **formato Parquet** en lugar de CSV.

En este post, analizaremos Apache Parquet, un formato extremadamente eficiente y muy utilizado en el ámbito de la ciencia de datos y la inteligencia artificial. El contenido del post esta orientado a explicar de forma clara en que consiste este formato, centrándonos en aspectos de alto nivel y utilizando SQL para abordar conceptos fundamentales.

1. Definición formal (TLDR)

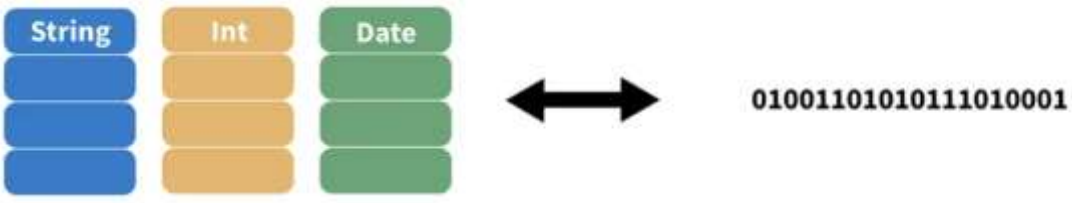
Apache parquet es un formato de archivo de código abierto que ***proporciona un almacenamiento eficiente y una velocidad de lectura rápida***. Utiliza un formato de almacenamiento híbrido que almacena secuencialmente columnas de datos, lo que produce un alto rendimiento al seleccionar y filtrar datos. Además incorpora soporte para los algoritmos de compresión mas populares (**Snappy**, **Gzip**, **LZO**). A su vez, incorpora algunos mecanismos inteligentes para ***reducir la lectura del archivo y la codificación de variables repetidas***.

Si es necesaria la lectura eficiente y rápida de un conjunto grande de datos, usar el ***formato Parquet*** puede ser una gran opción.

2. El Problema de Almacenamiento de Datos

Supongamos que somos ingenieros de datos y estamos buscando crear una arquitectura de datos que nos permita realizar procesos analíticos en línea (**OLAP**), que son operaciones de consulta muy concretas orientadas al análisis de datos.

Entonces, ¿cómo debemos almacenar nuestros datos en disco?



Bueno, hay muchos aspectos que se deben tener en cuenta, pero para trabajar en procesos OLAP, debemos considerar básicamente los dos mas importantes:

- ***Velocidad de lectura***: Como de rápido vamos a poder acceder y decodificar la información relevante que tenemos almacenada en ficheros binarios.
- ***Tamaño en disco***: El espacio que va a ocupar nuestro archivo estando en formato binario.

Hay que tener en cuenta que hay otras medidas que afectan directamente al resultado final en el proceso de almacenamiento de archivos en disco, como por ejemplo la velocidad de escritura o el soporte para metadatos, pero de momento tomaremos las dos anteriores como las más importantes.

Entonces, ¿cual es el rendimiento del ***formato Parquet*** en comparación al formato CSV? ***La realidad es que se necesita un 87% menos de espacio y las consultas son aproximadamente 34 veces más rápidas.***

3. Características clave del formato Parquet

¿Por que el ***formato Parquet*** es más eficiente que CSV y otros formatos de archivo que durante años se han utilizado en el campo de la ciencia de datos?

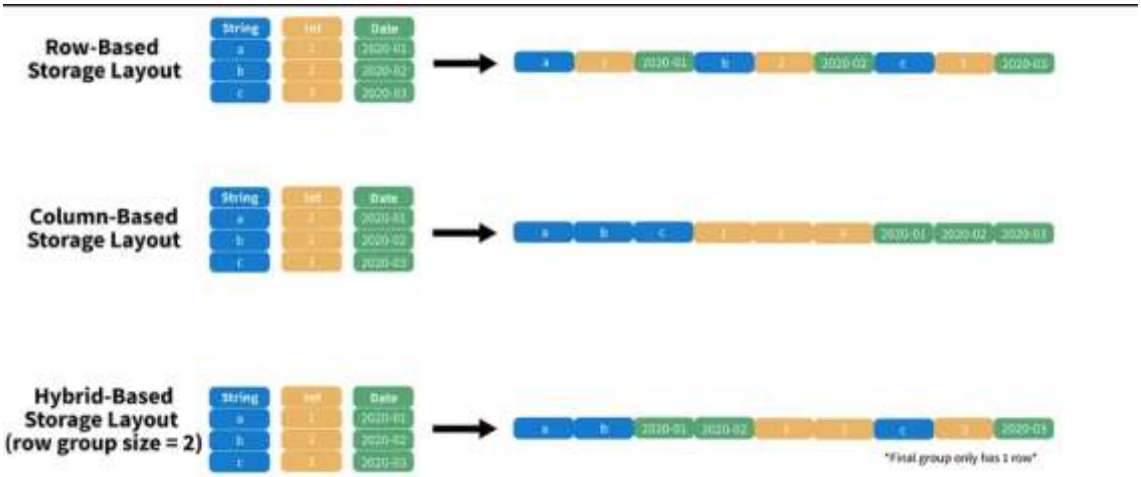
La primera respuesta está en su ***diseño interno para el almacenamiento de los datos…***

3.1 Almacenamiento híbrido de datos

Cuando queremos convertir una tabla de 2 dimensiones en una secuencia de 1’s y 0’s, debemos pensar detenidamente sobre la estructura óptima para almacenar estos datos.

¿Debería almacenarse la primera columna, luego la segunda, luego la tercera? ¿O deberíamos almacenar filas secuencialmente?

Tradicionalmente hay tres diseños de tipos de almacenamiento que convierten nuestra tabla de 2 dimensiones a 1 dimensión:



- ***Basado en filas (Row-based-layout)***: Almacenamiento secuencial de las filas (CSV).
- ***Basado en columnas (Column-based storage layout)***: Almacenamiento secuencial de las columnas (ORC).
- ***Basado en un formato híbrido (Hybrid-Based storage-layout)***: Almacenamiento secuencial de partes de columnas (Parquet).

Para los flujos de trabajo OLAP, los diseños híbridos son realmente efectivos porque admiten ***proyecciones y predicados***.

Proyección: es el proceso de selección de columnas y ***se puede considerar como la declaración SELECT en una consulta SQL***. La proyección es mucho más eficiente en un diseño basado en columnas. Por ejemplo, si queremos leer la primera columna de una tabla usando un diseño basado en columnas, únicamente hay que leer los primeros n índices de nuestro archivo binario, deserializarlos y extraer la información. Esto es algo muy eficiente.

Predicado: es el criterio utilizado para seleccionar filas y ***se puede considerar como la cláusula WHERE en una consulta SQL***. Los predicados son mas eficientes con el almacenamiento basado en filas. Si queremos todas las filas por algún criterio, como por ejemplo, obtener el todos los registros que cumplen la condición ***int >= 3***, podemos ordenar nuestra tabla de forma descendente, escanear todas las filas que no cumplan nuestro criterio y devolver todas las filas por encima de la última fila que no cumple el criterio.

En ambos escenarios, estamos intentando que el **sistema procese la menor cantidad de datos posible**. Por este motivo, un diseño de almacenamiento híbrido proporciona el equilibrio necesario entre un formato de archivo basado en filas y otro en columnas.

Antes de continuar, es importante dejar claro que el ***formato Parquet*** a menudo se describe como un formato columnar. Sin embargo, debido al hecho de que almacena partes de columnas, podemos afirmar que ***Parquet está basado en un diseño de almacenamiento híbrido***.

3.2 Metadatos en Parquet

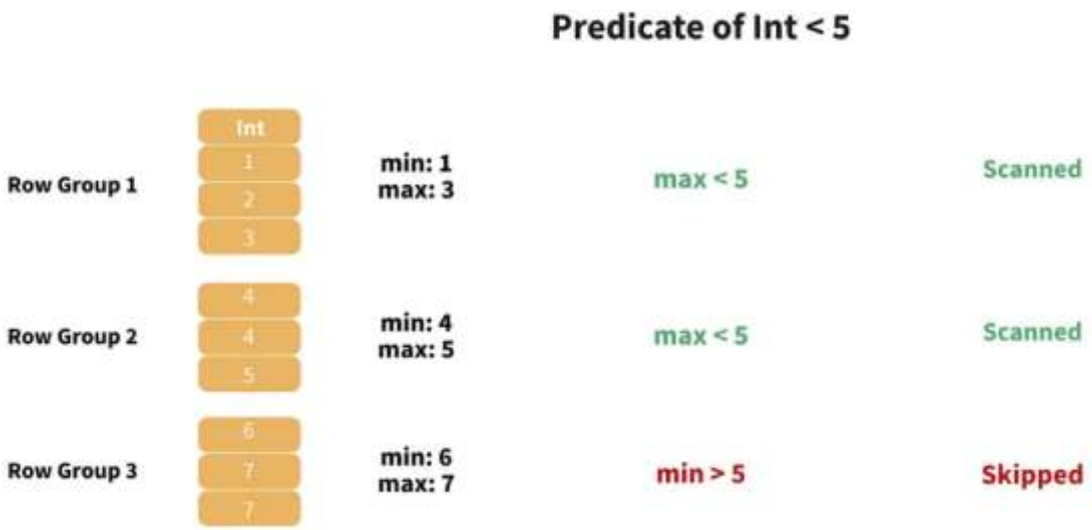
Parquet usa los metadatos para omitir la lectura de cierta parte de los datos de acuerdo con nuestro predicado.

En el ejemplo anterior, podemos ver que nuestro "***row group size***" es de tamaño 2, lo que significa que estamos almacenando 2 filas de la primera columna, 2 filas de la segunda, 2 filas de la tercera, y así hasta el final.

Cuando nos quedamos sin columnas, pasamos al siguiente conjunto de filas. Como sólo tenemos 3 filas en la tabla anterior el "***row group size***" final solo tendrá 1 fila.

Ahora supongamos que tenemos almacenadas 100.000 filas sin "***row groups***" de 2. Si queremos encontrar todas las filas donde una columna del tipo entero cumple la condición **int < 5** (es decir, un predicado de igualdad), el peor de los casos implicaría escanear todas las filas de la tabla.

Parquet realizaría ese filtro de la siguiente manera:



Como vemos, Parquet realiza un filtro inteligente de los datos, almacenando los valores máximo y mínimo para cada grupo de filas en los metadatos, ***lo que permite omitir grupos completos de filas***.

Por si no fuera suficiente, Parquet puede ***almacenar los datos en un conjunto de archivos con extensión ".parquet"***, lo que le permite, usando los metadatos, determinar si un fichero concreto debe leerse.

Al incluir metadatos, podemos omitir la lectura de ciertos datos y aumentar la velocidad de lectura.

3.3 Estructura de un fichero en formato Parquet

Como he mencionado, Parquet puede escribir muchos ficheros con extensión ***".parquet"***. En conjuntos de datos pequeños, esto es un problema ya que probablemente será necesario reparticionar los datos antes de poder almacenarlos. Sin embargo, en conjuntos de datos muy grandes, reparticionar el conjunto de datos en múltiples ficheros puede aumentar el rendimiento.

En general, el ***formato parquet*** sigue la siguiente estructura:

Directorio base > Ficheros Parquet > Grupo de filas > Columnas > Páginas de datos

Como podemos ver, existe un directorio base donde donde se almacenan todos los ficheros ***".parquet"***. Cada uno de estos ficheros contiene una partición de nuestros datos. **Un único fichero parquet se compone de muchos grupos de filas y un grupo de una fila contiene muchas columnas**. Finalmente, dentro de nuestras columnas están las páginas de datos, que en realidad contienen los datos sin procesar y metadatos relevantes.

Podemos ver una representación simplificada de un fichero .parquet:

Si quieres ampliar los detalles, aquí tienes la **documentación**.

3.4 Manejo de valores duplicados

Parquet evita los valores duplicados aplicando el uso de la ***Codificación de longitud de ejecución (RLE)***

Para entender el concepto, pondré un ejemplo. Supongamos que tenemos una columna con 10.000.000 de valores, pero todos los valores son 0. Para almacenar esta información, solo necesitamos 2 números: 0 y 10.000.000, es decir, ***el valor y el número de veces que se repite***.

La codificación RLE hace precisamente eso. Cuando se encuentran muchos valores duplicados, Parquet puede codificar esa información como una tupla (valor, recuento). De esta forma, nos ahorramos el almacenamiento de 9.999.998 números.

3.5 Manejo de tipos de datos muy grandes

Parquet usa la ***codificación de diccionarios*** y el ***Bit-Packing***.

Veamos un ejemplo. Supongamos que tenemos una columna que contiene nombres de países, algunos de los cuales son muy largos. Si quisiéramos almacenar "La República Democrática del Congo", necesitaríamos una columna del tipo "***String***" que pueda almacenar al menos 32 caracteres.

El uso de la **codificación de diccionarios** reemplaza cada valor en nuestra columna con un número entero pequeño y almacena esta asignación en la página de datos usando metadatos. En el disco, nuestros valores codificados están llenos de bits para ocupar la menor cantidad de espacio posible, pero cuando leemos los datos, podemos convertir nuestra columna en su valor original.

3.5 Uso de filtros complejos en los datos

Parquet puede manejar filtros complejos en los datos usando "***Projection and Predicate Pushdown***"

Esto es muy similar a la ejecución de filtros complejos un entorno con **Apache Spark**. En Spark podemos evitar leer toda una tabla entera en memoria usando el "***Projection and Predicate Pushdown***".

Las operaciones en Spark son evaluadas de forma "***lazy***", y únicamente son ejecutadas cuando se indica a través de una operación de terminación. Es en ese momento cuando Spark carga el resultado en memoria.

De esta forma, si solo queremos unas pocas columnas de grupo de filas, no tenemos que leer la tabla completa en memoria ya que es filtrada durante la operación de lectura.

Parquet usa un **sistema similar para realizar filtros complejos en los datos**. En la sección de enlaces y vídeos de interés hay un vídeo explicando el proceso.

En la sección de artículos y vídeos de interés he puesto el enlace a un vídeo que explica este concepto perfectamente.

4. Ejemplo práctico de uso: CSV vs Parquet

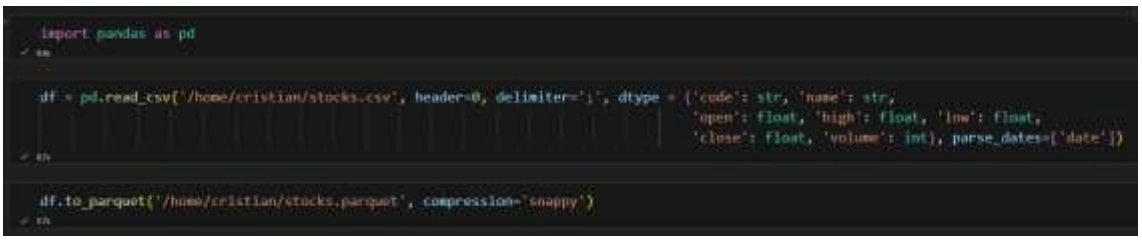
Para examinar todo lo aprendido, he obtenido un fichero CSV llamado **stocks.csv** que contiene los datos de cotización de empresas del **NASDAQ** entre 2010 y 2020.

Con 10 años de datos, el fichero CSV tiene un tamaño aproximado de 1 Gb. El archivo **CSV** tiene la siguiente estructura:

```
"date";"code";"name";"open";"high";"low";"close";"volume";"unadjusted_close"
2015-09-30;FLWS;"1-800 FLOWERS.COM";8.96000;9.25000;8.94000;9.10000;541400;0.00000
2014-06-16;FLWS;"1-800 FLOWERS.COM";5.67000;5.75000;5.60000;5.68000;82100;0.00000
2012-03-15;FLWS;"1-800 FLOWERS.COM";2.98000;3.00000;2.94000;3.00000;51600;0.00000
2015-07-23;FLWS;"1-800 FLOWERS.COM";10.78000;10.84000;10.32000;10.39000;261800;0.00000
2015-08-17;FLWS;"1-800 FLOWERS.COM";9.49000;9.76000;9.41000;9.74000;154800;0.00000
2015-08-18;FLWS;"1-800 FLOWERS.COM";9.73000;9.80000;9.62000;9.74000;220400;0.00000
2015-08-19;FLWS;"1-800 FLOWERS.COM";9.64000;9.90000;9.47000;9.73000;262900;0.00000
2015-08-20;FLWS;"1-800 FLOWERS.COM";9.61000;9.69000;9.45000;9.45000;308300;0.00000
2015-08-25;FLWS;"1-800 FLOWERS.COM";9.63000;9.63000;9.14000;9.38000;314100;0.00000
2015-08-26;FLWS;"1-800 FLOWERS.COM";9.62000;9.71000;9.06000;9.60000;611500;0.00000
2015-08-27;FLWS;"1-800 FLOWERS.COM";8.49000;9.03000;7.96000;8.43000;2067700;0.00000
2015-08-28;FLWS;"1-800 FLOWERS.COM";8.37000;8.92000;8.34000;8.91000;1026900;0.00000
2015-08-31;FLWS;"1-800 FLOWERS.COM";8.83000;8.87000;8.26000;8.39000;962400;0.00000
2015-09-04;FLWS;"1-800 FLOWERS.COM";8.68000;8.79000;8.55000;8.56000;367200;0.00000
2010-01-08;FLWS;"1-800 FLOWERS.COM";2.54000;2.64000;2.50000;2.54000;76200;0.00000
2010-01-01;FLWS;"1-800 FLOWERS.COM";2.65000;2.65000;2.65000;2.65000;0;0.00000
2010-01-04;FLWS;"1-800 FLOWERS.COM";2.69000;2.74000;2.51000;2.61000;152500;0.00000
2010-01-05;FLWS;"1-800 FLOWERS.COM";2.60000;2.63000;2.50000;2.52000;106500;0.00000
2010-01-07;FLWS;"1-800 FLOWERS.COM";2.57000;2.69000;2.50000;2.56000;98000;0.00000
2010-01-11;FLWS;"1-800 FLOWERS.COM";2.55000;2.62000;2.49000;2.54000;124600;0.00000
```

Para este ejemplo práctico voy a usar Python y las librerías ***Pandas*** y ***fastparquet***.

Para trabajar con el ***formato Parquet*** lo primero que hemos de hacer es convertir ese fichero CSV a ***formato Parquet***. Para ello usaremos las dos librerías anteriormente mencionadas de la siguiente forma:



Como se puede ver en el notebook de Jupyter, al usar la función ***read_csv*** de pandas he especificado el tipo de dato de cada una de las columnas. Esto es muy importante

para que la exportación a ***formato Parquet*** se realice con los tipos adecuados y el formato sepa optimizarlos correctamente.

La exportación a Parquet se ha realizado correctamente y se ha especificado el uso de compresión en el formato ***Snappy***

El resultado de la exportación es:

De momento, podemos ver que una vez realizada la conversión a ***formato parquet*** el almacenamiento se ha ***reducido en un 70% del espacio original (De 1Gb a aprox. 300 Mb)***. Esto es fruto de la aplicación de los diferentes conceptos que hemos visto en las anteriores secciones.

Usando la librería ***fastparquet*** vamos a analizar el fichero ".parquet" que hemos generado. Primero vamos a ver las columnas y los tipos que ha generado el formato:

```
from fastparquet import ParquetFile
pf = ParquetFile('/home/cristian/stocks.parquet')
```

```
pf.columns
```

```
['date',
 'code',
 'name',
 'open',
 'high',
 'low',
 'close',
 'volume',
 'unadjusted_close']
```

```
pf.dtypes
```

```
OrderedDict([('date', 'datetime64[ns]'),
             ('code', dtype('O')),
             ('name', dtype('O')),
             ('open', dtype('float64')),
             ('high', dtype('float64')),
             ('low', dtype('float64')),
             ('close', dtype('float64')),
             ('volume', dtype('int64')),
             ('unadjusted_close', dtype('float64'))])
```

Como podemos ver, tanto las columnas como el tipo de datos se han inferido correctamente al exportar el "dataframe" desde Pandas.

En lo referente a la velocidad de carga de un "dataframe" desde ***CSV*** o ***Parquet*** podemos ver las diferencias:

En CSV tarda unos 10,8 segundos.

Usando ***Parquet*** vemos una notable diferencia:

En el formato ***Parquet*** la carga del "dataframe" es un 61% más rápida.

Conclusión

Parquet es un formato de fichero óptimo y muy utilizado en el mundo de la ciencia de datos. Es muy efectivo reduciendo las lecturas necesarias en una tabla y comprimiendo los datos.