Operaciones básicas en Spark

- Spark opera con colecciones **inmutables y distribuidas** de elementos, manipulándolos en paralelo
  - API estructurada: DataFrames y DataSets
  - API de bajo nivel: RDDs (ya obsoleto ya que el API estructurada es más eficiente y más de alto nivel)
- Operaciones sobre estas colecciones
  - Creación
  - Transformaciones (ordenación, filtrado, etc.)
  - Realización acciones para obtener resultados
- Spark automáticamente distribuye los datos y paraleliza las operaciones

Ejemplo: creación de un DataFrame a partir de un fichero CSV

En este ejemplo, Spark infiere el esquema de los datos de forma automática

- Es preferible especificar el esquema de forma explícita, como veremos más adelante

También se especifica que la primera línea es la cabecera.

In [ ]:
```
%%sh
wget -q "https://raw.githubusercontent.com/dsevilla/tcdm-public/24-25/datos/2015-summary.csv"
ls -lh 2015-summary.csv
head 2015-summary.csv
```

In [ ]:
```
%pip install pyspark
```

In [ ]:
```
from pyspark import SparkContext
from pyspark.sql import SparkSession

# Creamos un objeto SparkSession (o lo obtenemos si ya está creado)
spark: SparkSession = SparkSession \
  .builder \
  .appName("Mi aplicacion") \
  .config("spark.alguna.opcion.de.configuracion", "algun-valor") \
  .master("local[*]") \
  .getOrCreate()

sc: SparkContext = spark.sparkContext
```

In [ ]:
```
from pyspark.sql.dataframe import DataFrame

datosVuelos2015: DataFrame = (spark
    .read
    .option("inferSchema", "true")
    .option("header", "true")
    .csv("2015-summary.csv"))
```

In [ ]:
```
datosVuelos2015.printSchema()
```

In [ ]:
```
datosVuelos2015.show()
print(datosVuelos2015.count())

assert(datosVuelos2015.count() == 256)
```

In [ ]:
```
datosVuelos2015.show(5)
```

Rows

Las filas de un DataFrame son objetos de tipo `Row`

- API de Row en Python: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Row.html

In [ ]:
```
# Obtenemos las dos primeras fila del DataFrame
from pyspark.sql.types import Row

rows1_2: list[Row] = datosVuelos2015.take(2)
print(rows1_2)
print(type(rows1_2))
print(type(rows1_2[0]))
```

In [ ]:
```
# Obtén la primera fila como un diccionario Python
```

```
print(rows1_2[0].asDict())
print(type(rows1_2[0].asDict()))

assert(type(rows1_2[0].asDict()) is dict)
```

Particiones

Spark divide las filas DataFrame en un conjunto de particiones

- El número de particiones por defecto es función del tamaño del cluster (número total de cores en todos los ejecutores) y del tamaño de los datos (número de bloques de los ficheros en HDFS)
- Para RDDs se puede especificar otro valor en el momento de crearlos
- También se puede modificar una vez creados

In [ ]:
```
print("Número de particiones: {0}"
    .format(datosVuelos2015.rdd.getNumPartitions()))

# Creo un nuevo DataFrame con 4 particiones
datosVuelos2015_4P: DataFrame = datosVuelos2015.repartition(4)
print("Número de particiones: {0}"
    .format(datosVuelos2015_4P.rdd.getNumPartitions()))

assert(datosVuelos2015_4P.rdd.getNumPartitions() == 4)
```

Transformaciones

Operaciones que transforman los datos

- No modifican los datos de origen (*inmutabilidad*)
- Se computan de forma “perezosa” (*lazyness*)

Dos tipos:

- Transformaciones *estrechas* (narrow)
  - Cada partición de entrada contribuye a una única partición de salida
  - No se modifica el número de particiones
  - Normalmente se realizan en memoria
- Transformaciones *anchas* (wide)
  - Cada partición de salida depende de varias (o todas) particiones de entrada
  - Suponen un barajado de datos
  - Pueden implicar un cambio en el número de particiones
  - Pueden suponer escrituras en disco

In [ ]:
```
# Ejemplo de una transformación narrow
datosVuelos2015_EEUU: DataFrame = datosVuelos2015\
    .replace("United States", "Estados Unidos")
```

In [ ]:
```
# Ejemplo de una transformación wide
datosVuelos2015_Ord: DataFrame = datosVuelos2015_EEUU\
    .sort("count", ascending=False)
datosVuelos2015_Ord.cache()
```

Acciones

Obtienen un resultado, forzando a que se realicen las transformaciones pendientes

- En el momento de disparar la *acción* se crea un *plan* con las transformaciones necesarias para obtener los datos solicitados
  - Se crea un Grafo Dirigido Acíclico (DAG) conectando las transformaciones
  - Spark optimiza ese grafo, para eliminar transformaciones innecesarias o unir las que sea posible
- Las acciones traducen el DAG en un plan de ejecución

Tipos de acciones

- Acciones para mostrar datos por consola
- Acciones para convertir datos Spark en datos del lenguaje
- Acciones para escribir datos a disco

In [ ]:
```
# Ejemplo de acciones
from pprint import pp

print("Número de filas en la tabla: {0}"
    .format(datosVuelos2015_Ord.count()))
```

```
pp(datosVuelos2015_Ord.take(3))

datosVuelos2015_Ord.show()
```