

Arquitecturas Cloud y Big Data



red.es

Centro de
Referencia Nacional
en Comercio Electrónico
y Marketing

CRN
Digital



"El FSE invierte en tu futuro"
Fondo Social Europeo

Índice

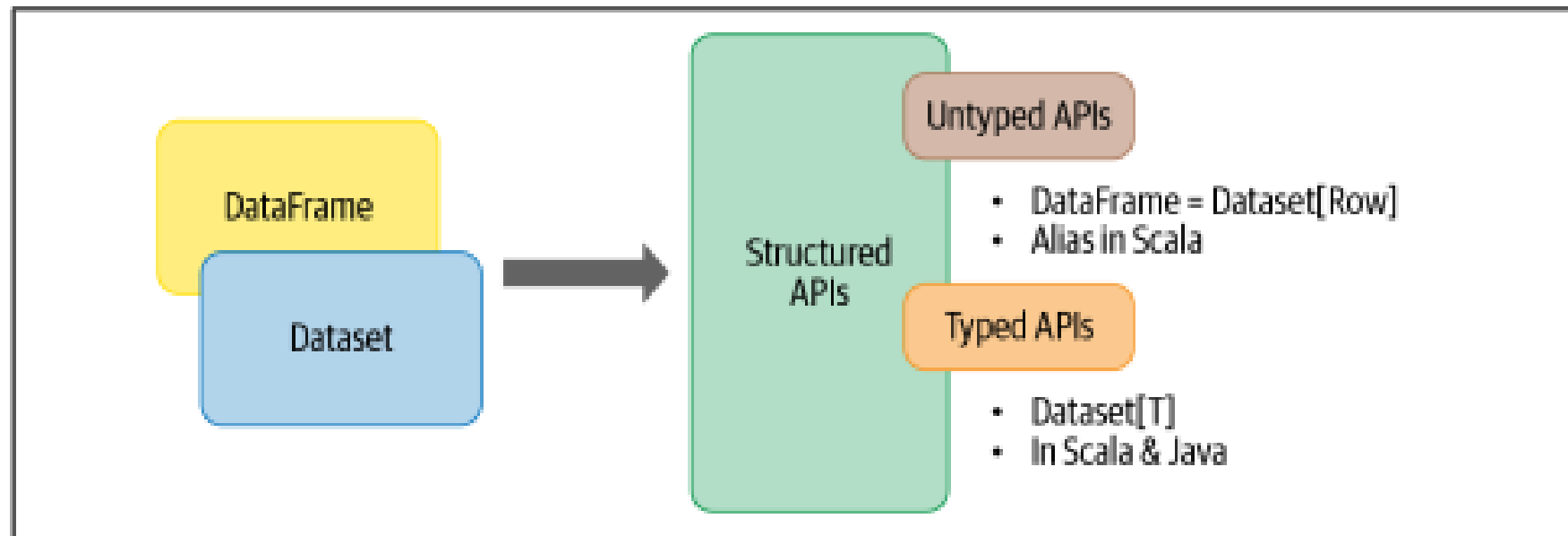
1. **PySpark Structured APIs: Dataframe**
2. **Transformaciones DataFrame: consulta de datos**
3. **Transformaciones DataFrame: modificar datos**

1. PySpark Structured APIs: Dataframes



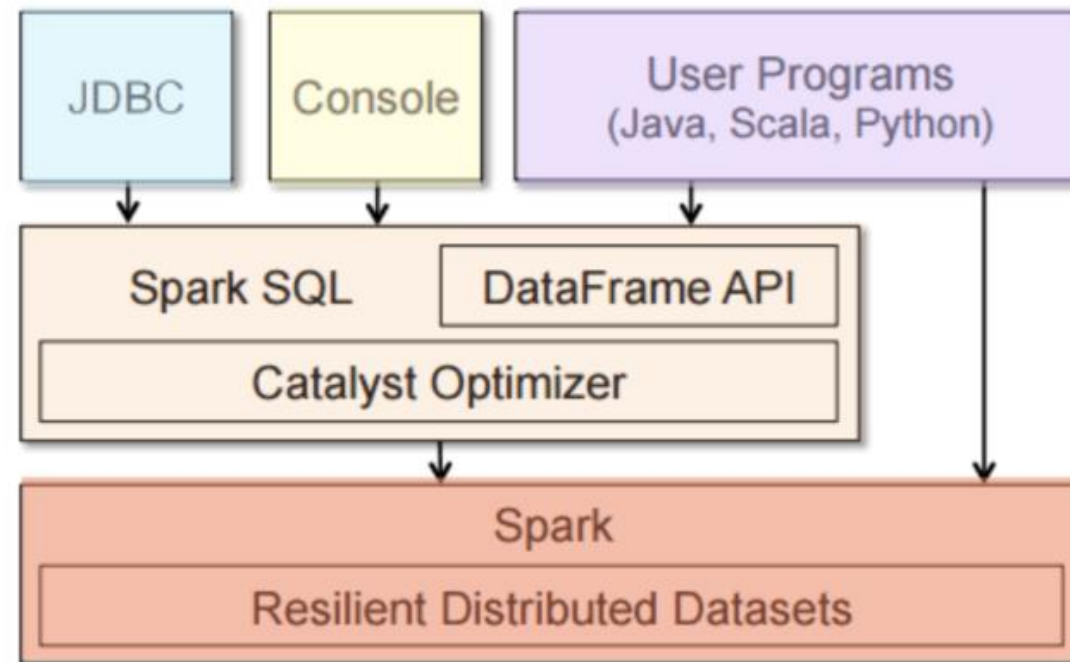
Structured APIs: Dataframes y Datasets

- Spark define unas APIs para datos estructurados (como “tablas”, colección de registros): Dataframes y Datasets.
- Se define una estructura (**schema**) para cada instancia de ellos.
- La diferencia es en la “comprobación” de ese esquema: Dataframes (runtime) frente a Datasets (compile time, **sólo Java y Scala, no los usaremos**)



DATAFRAMES

- Los Spark DataFrames son como tablas distribuidas en memoria con **columnas con nombre** y **esquemas**, donde cada columna tiene un tipo de datos específico: entero, cadena, real, array, fecha, timestamp, etc.
- En ejecución Spark las “compila” a la API de bajo nivel que ya conocemos: RDDs



Schemas: tipos de datos columnas

Data type	Value assigned in Python	API to instantiate
ByteType	int	DataTypes.ByteType
ShortType	int	DataTypes.ShortType
IntegerType	int	DataTypes.IntegerType
LongType	int	DataTypes.LongType
FloatType	float	DataTypes.FloatType
DoubleType	float	DataTypes.DoubleType
StringType	str	DataTypes.StringType
BooleanType	bool	DataTypes.BooleanType
DecimalType	decimal.Decimal	DecimalType

Data type	Value assigned in Python	API to instantiate
BinaryType	bytearray	BinaryType()
TimestampType	datetime.datetime	TimestampType()
DateType	datetime.date	DateType()
ArrayType	List, tuple, or array	ArrayType(dataType, [nullable])
MapType	dict	MapType(keyType, valueType, [nullable])
StructType	List or tuple	StructType([fields])
StructField	A value type corresponding to the type of this field	StructField(name, dataType, [nullable])

CREAR UN DATAFRAME (I)

- Datos + esquema . El esquema lo podemos definir (explícito, lo más aconsejado) o **inferir** desde los datos **en el caso de lectura de una fuente**.

```
datos = [(None, 'Smith', '36636', 'M', 3500),  
         ('Michael', 'Rose', '40288', 'M', 4750),  
         ('Robert', 'Williams', '42114', 'M', None),  
         ('Maria', 'Jones', '39192', 'F', 4000)]
```

Define esquema como lista de columnas

Define columna

```
from pyspark.sql.types import StructType, StructField, StringType,  
IntegerType
```

```
esquema = StructType([  
    StructField('firstname', StringType(), True),  
    StructField('lastname', StringType(), False),  
    StructField('id', StringType(), False),  
    StructField('gender', StringType(), True),  
    StructField('salary', IntegerType(), True)
```

Nombre columna

Tipo

Se permite valor NULO

```
] )
```

CREAR UN DATAFRAME (II)

- Recordar, en nuestro entorno interactivo ya están instanciados “SparkSession” como `<spark>`, y “SparkSession.sparkContext” como `<sc>`, las utilizamos directamente

```
df = spark.createDataFrame(data=datos, schema=esquema)

df.printSchema()
```

- **“printShema()”**: muestra esquema del Dataframe
- **“show()”**: ACCIÓN, muestra el contenido (se puede especificar nº registros entre paréntesis, al igual que si se recorta o no la salida por pantalla)

```
df.show(truncate=False)
```


DATAFRAME: objeto “Row”

- En Spark se define un objeto **Row** (fila) como una colección ordenada de campos. Cada uno de sus campos formaría parte de una columna del DF.
- Se puede acceder de forma individual a los campos. Siguiendo el ejemplo de los empleados al crear un dataframe:

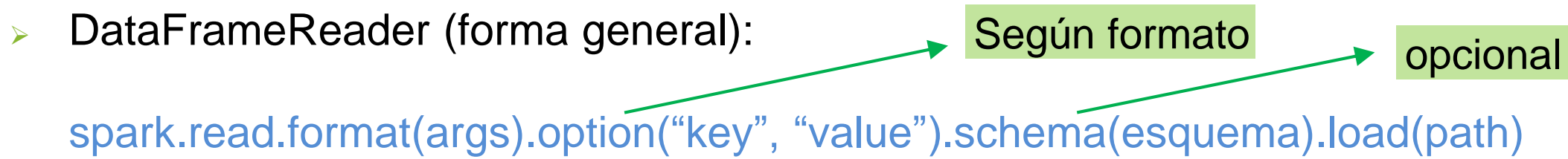
```
from pyspark.sql import Row

empleado_1 = Row("James", "Smith", "36636", "M", 3500)

print('Este empleado se llama: ', empleado_1[0])
```

```
datos = [Row("James", "Smith", "36636", "M", 3500) ,
          Row("Michael", "Rose", "40288", "M", 4750) ,
          Row("Robert", "Williams", "42114", "M", 4200) ,
          Row("Maria", "Jones", "39192", "F", 4000)
        ]
```

Leer DATAFRAME desde un archivo (I)

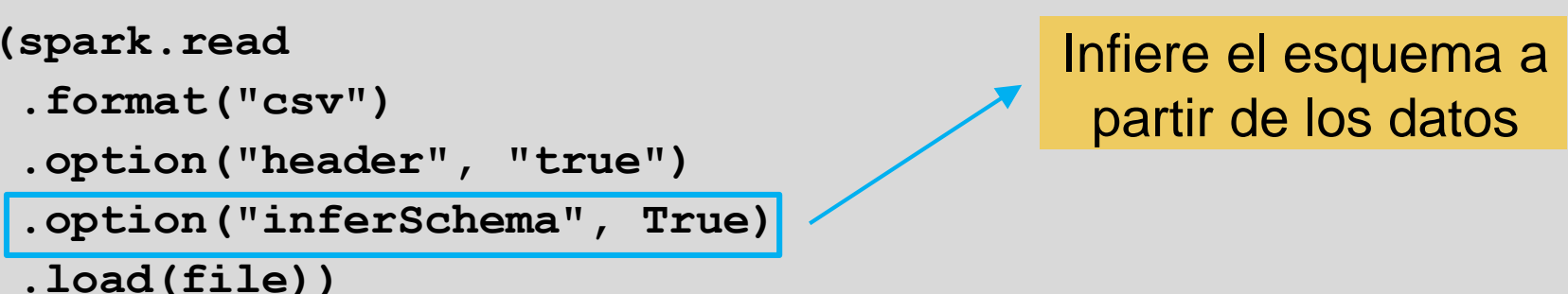
- Spark SQL admite una gran variedad de fuentes de datos, proporcionando un conjunto de métodos comunes para leer (**DataFrameReader**) y escribir (**DataFrameWriter**) datos en y desde estas fuentes.
- DataFrameReader (forma general):


```
spark.read.format(args).option("key", "value").schema(esquema).load(path)
```
- Ejemplo lectura archivo: **sales.csv**, con opción "inferSchema"

```
file = 'dbfs:/FileStore/shared_uploads/edurf.cld@gmail.com/sales.csv'

sales_df = (spark.read
            .format("csv")
            .option("header", "true")
            .option("inferSchema", True)
            .load(file))

sales_df.printSchema()
```



Leer DATAFRAME desde un archivo (II)

➤ Ejemplo lectura archivo: **persons.json**, especificando esquema

```
from pyspark.sql.types import IntegerType, StringType, FloatType, ArrayType,
DateType, BooleanType

persons_schema = StructType([
    StructField('id', IntegerType(), True),
    StructField('first_name', StringType(), True),
    StructField('last_name', StringType(), True),
    StructField('fav_movies', ArrayType(StringType()), True),
    StructField('salary', FloatType(), True),
    StructField('image_url', StringType(), True),
    StructField('date_of_birth', DateType(), True),
    StructField('active', BooleanType(), True)
])
```

```
file = 'dbfs:/FileStore/shared_uploads/edurf.cld@gmail.com/persons.json'

persons_df = (spark.read.format('json')
              .option('multiline', True)
              .schema(persons_schema)
              .load(file)
              )
persons_df.show()
```

especificamos el esquema

2.Transformaciones

Dataframe:

Consulta datos



Transformaciones dataframe: SELECT

- **SELECT**: devuelve dataframe con la(s) columna(s) y/o expresiones especificadas.

Podemos escribir directamente el/los nombre(s) de las columnas (cadena)

```
sales_df.select('Order_ID', 'Item_Type', 'Units_Sold', 'Unit_Price', 'Country')
.show(10, truncate=False)
```

RECORDAR, necesitamos **ACCIÓN** “**show**” (evaluación perezosa)

- Para expresiones más complejas podemos hacer uso de “**col**” (columna), “**expr**” (expresión), “**alias**”

```
from pyspark.sql.functions import col, expr

sales_df.select(col('Order_ID'), col('Item_Type'), expr("Units_Sold *
Unit_Price as TOTAL_PRICE")).show(10)
```

```
sales_df.select(col('Order_ID'), col('Item_Type'), (col('Units_Sold') *
col('Unit_Price')).alias('TOTAL_PRICE')).show(10)
```

- NO podemos utilizar ambas formas (solo cadenas frente al uso col, expr) mismo Select

Transformaciones dataframe: FILTER/WHERE

- **FILTER/WHERE**: devuelve dataframe con los registros que cumplan la condición expresada. Se obtiene el mismo resultado con ambas (WHERE por semejanza SQL)

```
(sales_df.filter((col('Region')== 'Europe') & (col('Country')== 'Spain'))  
          .select(col('Order_ID'), col('Country'),  
col('Item_Type'), expr("Units_Sold * Unit_Price as TOTAL_PRICE"))).show(5)
```

Debemos emplear como operadores booleanos:

- “&” como “AND” - “|” como “OR” (AltGR+1) - “~” como “NOT” (AltGR+4+ESP)

- Si “encadenamos” (notación punto) dos where/filter, el efecto es el del operador “&”

```
(sales_df.where(col('Region')== 'Europe').where(col('Country')== 'Spain')  
          .select(col('Order_ID'), col('Country'),  
col('Item_Type'), expr("Units_Sold * Unit_Price as TOTAL_PRICE"))).show(5)
```

Transformaciones dataframe: ORDERBY

- **ORDERBY**: Devuelve Dataframe con los valores ordenados por la(s) columna(s) especificadas. Podemos usar '**asc()**' (por defecto) y '**desc()**' para especificar orden

```
(sales_df
.select(col('Order_ID'), col('Country'), col('Item_Type'),
col('Units_Sold'))
.orderBy(col('Units_Sold').desc())) .show(10)
```

```
(sales_df.select(col('Order_ID'),col('Country'),col('Item_Type'),col('Units_Sold'))
.orderBy(col('Units_Sold').desc(),col('Country').asc())) .show(20,truncate=False)
```

Para tratar con valores nulos existen las opciones '**asc_nulls_first()**', '**desc_nulls_first()**', '**asc_nulls_last()**', '**desc_nulls_last()**'

```
(sales_df.select('Region','Country')
.orderBy(col('Region').asc_nulls_first())) .show()
```

Transformaciones dataframe: DISTINCT, LIMIT

- **DISTINCT**: Devuelve los valores únicos (distintos) del dataframe (por ejemplo para encontrar valores únicos de una columna, contarlos)

```
print(sales_df.select('Region').distinct().count())  
  
sales_df.select('Region').distinct().show(truncate=False)
```

- **LIMIT**: Restringe el número de registros del dataframe a devolver al especificado entre paréntesis

```
(sales_df.select(col('Order_ID'), col('Country'), col('Item_Type'), col('Units_Sold'))  
.orderBy(col('Units_Sold').desc(), col('Country').asc()).limit(20)).count()
```

NO confundir con el uso de la acción “show()”

- “**show(10)**” devuelve por pantalla los 10 primeros registros del dataframe completo
- “**limit(10)**” “recorta” el dataframe a los 10 primeros

EJERCICIO: archivo Sales

➤ **Realizar la siguiente CONSULTA:**

Devolver los campos producto, unidades vendidas, fechas de pedido y envío, de las ventas de la Zona Logística de Asia, ordenadas por país. Sólo nos interesan los 10 primeros.

3.Transformaciones

Dataframe:

Modificar datos



AÑADIR Y RENOMBRAR COLUMNA: WITHCOLUMN

- **withColumn()**: Añade una nueva columna al Dataframe, por ejemplo un campo calculado.

```
from pyspark.sql.functions import lit
```

```
# VALOR DETERMINADO
```

```
sales_df.withColumn("Sent", lit(False)).show(5)
```

Nombre nueva columna

VALOR

```
# CAMPO CALCULADO
```

```
sales_df.withColumn("Total_Price", expr("Units_Sold *  
Unit_Price")).show(5)
```

- **withColumnRenamed()**: cambia nombre de una columna

```
sales_df.withColumnRenamed("Region", "Logistics_Area").show(5)
```

BORRAR COLUMNA(S): DROP

- **drop()**: elimina una columna o columnas del Dataframe.

Por ejemplo en una dataframe resumido nos puede interesar solo el número de unidades y el precio total, no el precio unitario. Se pueden indicar varias columnas, una tras otra.

```
resumen_df = sales_df.withColumn("Total_Price", expr("Units_Sold *  
Unit_Price"))  
  
resumen_df.show(10)
```

```
resumen_df = resumen_df.drop('Unit_Price', 'Region')  
  
resumen_df.printSchema()
```

ELIMINAR FILAS CON NULOS: dropna()

➤ dropna(how='any', thresh=None, subset=None):

Elimina filas con valores nulos, según valores parámetros.

- how: 'any', 'all'

- thresh: nº de valores válidos (no nulos) que debe tener como mínimo la fila, si no se elimina

- subset: aplicar a sólo a una(s) columna(s)

```
df.show()

# eliminamos aquellos con salario nulo
not_null_df = df.dropna(subset='salary')

not_null_df.show()
```

OTROS: cambiar tipo <cast()>; tratamiento de cadenas

- **cast()**: junto con <withColumn()> “cambia” el tipo de una columna.

```
sales_df.withColumn('Order_ID',  
col('Order_ID').cast('string')).printSchema()
```

- **Cadenas, pasar a MAY/min:** upper(), lower()

```
from pyspark.sql.functions import upper, lower  
  
corregido1 = df.withColumn('firstname', upper(col('firstname')))  
  
corregido1.show()
```

- **Cadenas, eliminar espacios:** ltrim(), rtrim(), trim()

```
from pyspark.sql.functions import ltrim, rtrim, trim  
  
corregido2 = df.withColumn('lastname', trim(col('lastname')))  
  
corregido2.show()
```

ALMACENAR DATAFRAME.

- Spark SQL admite gran variedad de formatos para almacenar

- DataFrameWriter (forma general):

Formato
almacenamiento

```
dataframe.write.format(args).mode(save_mode)
```

```
.option("key", "value").save(path)
```

- Save mode: append / overwrite / errorIfExists / ignore

```
path = 'dbfs:/FileStore/shared_uploads/SUSTITUIR_USUARIO/sales'
(sales_df.write
    .format("parquet")
    .mode("overwrite")
    .option("compression", "snappy")
    .save(path))
```

```
%fs ls 'dbfs:/FileStore/shared_uploads/SUSTITUIR_USUARIO/sales'
```

EJERCICIO: archivo Sales

- Partir del Dataframe “Sales” original (desde cero)
- Simulación ETL sencilla.
 - Crear columna descuento fijo (idea: si expresamos el **descuento** como proporción, lo podemos utilizar directamente para calcular el precio, multiplicando)

Precio: 200 → descuento 25% → $1 - 0,25 = \underline{0,75}$ → precio final = **0,75** * precio
 - Crear columna precio total (precio * unidades * descuento). Eliminar después columna precio unitario y descuento
 - Pasar el campo Región a MAYÚSCULAS. Cambiar el nombre del campo a “Logist_Area”



red.es



UNIÓN EUROPEA

"El FSE invierte en tu futuro"

Fondo Social Europeo

