

SESIÓN PROCESAMIENTO DE DATOS

CONTENIDOS:

- Spark sql
 - Qué es spark sql.
 - Características de spark sql.
 - Qué es una sesión spark.
- Dataframes
 - Qué es un dataframe spark
 - Cómo funciona.
 - Métodos de creación de un dataframe.
 - Métodos de exploración de un dataframe.
 - Uso de dataframes para consultas.
 - Comparación entre dataframe y dataset.
- Funciones definidas por el usuario.
- Fuentes y destinos de datos: interoperación de los rdd.
- Formatos de archivo json y parquet.
- Lectura y escritura desde/hacia distintas fuentes.

SPARK SQL

¿Qué es Spark SQL?

Spark SQL es un módulo de Apache Spark diseñado para trabajar con datos estructurados. Permite ejecutar consultas SQL dentro de un entorno Spark y proporciona una API para trabajar con datos en un formato estructurado, como DataFrames y Datasets.

Características de Spark SQL

- **Compatibilidad con SQL estándar:** Permite ejecutar consultas SQL directamente sobre los datos.
- **Uso de DataFrames y Datasets:** Proporciona estructuras de datos optimizadas para manejar grandes volúmenes de información.

- **Optimización con Catalyst:** Incluye un optimizador de consultas llamado Catalyst que mejora el rendimiento de las consultas SQL.
- **Compatibilidad con múltiples fuentes de datos:** Puede leer y escribir datos en formatos como JSON, Parquet, ORC, Avro, y conectarse con bases de datos como MySQL y PostgreSQL.
- **Interoperabilidad con RDDs:** Spark SQL permite convertir RDDs en DataFrames para aprovechar las optimizaciones de procesamiento.



Ilustración 1 Spark SQL

¿Qué es una sesión Spark?

Una sesión Spark (*SparkSession*) es el punto de entrada principal para interactuar con Apache Spark y su módulo Spark SQL.

Permite:


- Crear y administrar DataFrames y Datasets.
- Leer y escribir datos en diferentes formatos y fuentes.
- Ejecutar consultas SQL.

Ejemplo de creación de una sesión Spark en Python:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("EjemploSparkSQL") \
    .getOrCreate()
```

Ilustración 2 Crear Sesión



Este código inicializa una sesión de Spark con un nombre específico. Si una sesión ya está activa, la reutiliza.

DATAFRAMES

¿Qué es un DataFrame en Spark?

Un DataFrame en Spark es una colección distribuida de datos organizados en columnas con nombres y tipos definidos, similar a una tabla en una base de datos o un DataFrame de pandas.

¿Cómo funciona un DataFrame en Spark?

Los DataFrames se basan en la ejecución diferida (*lazy evaluation*), lo que significa que las operaciones sobre ellos no se ejecutan inmediatamente, sino que Spark optimiza y ejecuta las acciones cuando es necesario. Esto mejora la eficiencia y el rendimiento.

Métodos de creación de un DataFrame

Los DataFrames pueden crearse de varias maneras:

1. A partir de una lista o diccionario en Python

```
from pyspark.sql import Row

data = [Row(id=1, nombre="Ana"), Row(id=2, nombre="Carlos")]
df = spark.createDataFrame(data)
df.show()
```

2. A partir de un archivo CSV, JSON o Parquet

```
df_csv = spark.read.csv("datos.csv", header=True, inferSchema=True)
df_json = spark.read.json("datos.json")
df_parquet = spark.read.parquet("datos.parquet")
```

3. A partir de un RDD

```
rdd = spark.sparkContext.parallelize([(1, "Ana"), (2, "Carlos")])  
df_rdd = spark.createDataFrame(rdd, ["id", "nombre"])
```

Métodos de exploración de un DataFrame

Una vez creado el DataFrame, se pueden aplicar varios métodos para explorar su contenido.

1. Mostrar datos

```
df.show() # Muestra las primeras filas  
df.show(5, truncate=False) # Muestra 5 filas sin truncar el contenido
```

2. Ver la estructura del DataFrame

```
df.printSchema() # Muestra el esquema de columnas y tipos de datos
```

3. Contar filas

```
df.count()
```

4. Describir los datos

```
df.describe().show()
```

Uso de DataFrames para consultas

Spark permite usar SQL sobre DataFrames, lo que facilita la manipulación de datos.

Ejemplo de consulta SQL en un DataFrame

```
df.createOrReplaceTempView("personas")
resultado = spark.sql("SELECT * FROM personas WHERE id > 1")
resultado.show()
```

Comparación entre DataFrame y Dataset

Característica	DataFrame	Dataset
Tipo de datos	Esquemático, basado en filas y columnas.	Tipo fuertemente tipado con soporte para operaciones funcionales.
Seguridad de tipos	No verifica tipos en tiempo de compilación.	Verifica tipos en tiempo de compilación (solo en Scala y Java).
API	Disponible en Python, Scala y Java.	Principalmente en Scala y Java.
Rendimiento	Optimizado con Catalyst.	Más rápido si se usan tipos definidos.

FUNCIONES DEFINIDAS POR EL USUARIO (UDFS)

Las UDFs (*User Defined Functions*) permiten definir funciones personalizadas que pueden aplicarse a DataFrames en Spark.

Ejemplo de creación de una UDF en Python

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def convertir_mayuscula(texto):
    return texto.upper()

udf_mayuscula = udf(convertir_mayuscula, StringType())
df_con_udf = df.withColumn("nombre_mayuscula", udf_mayuscula(df["nombre"]))
df_con_udf.show()
```

Ilustración 3 Creación de una UDF

FUENTES Y DESTINOS DE DATOS: INTEROPERACIÓN DE LOS RDDS

Spark permite transformar datos entre diferentes estructuras como RDDs, DataFrames y Datasets.

Conversión de un RDD a DataFrame

```
df_desde_rdd = rdd.toDF(["id", "nombre"])
df_desde_rdd.show()
```

Ilustración 4 Conversión de un RDD a DataFrame

Conversión de un DataFrame a RDD

```
rdd_desde_df = df.rdd
print(rdd_desde_df.collect())
```

Ilustración 5 Conversión de DataFrame a RDD

FORMATOS DE ARCHIVO: JSON Y PARQUET

JSON

- Formato ligero y basado en texto.
- Auto-descriptivo y legible.
- Compatible con la mayoría de los lenguajes de programación.

Ejemplo de lectura y escritura en JSON:

```
df_json = spark.read.json("datos.json")
df_json.write.json("salida.json")
```

Ilustración 6 Lectura y escritura de JSON

Parquet

- Formato de almacenamiento columnar.
- Optimizado para consultas rápidas y compresión eficiente.
- Recomendado para grandes volúmenes de datos.

Ejemplo de lectura y escritura en Parquet:

```
df_parquet = spark.read.parquet("datos.parquet")
df_parquet.write.parquet("salida.parquet")
```

Ilustración 7 Lectura y escritura de Parquet

LECTURA Y ESCRITURA DESDE/HACIA DISTINTAS FUENTES

Spark SQL permite trabajar con múltiples fuentes de datos.

Lectura desde una base de datos MySQL

```
df_mysql = spark.read.format("jdbc") \
    .option("url", "jdbc:mysql://localhost:3306/mi_base") \
    .option("dbtable", "usuarios") \
    .option("user", "root") \
    .option("password", "1234") \
    .load()
df_mysql.show()
```

Ilustración 8 Lectura desde una base de datos

Escritura en una base de datos PostgreSQL

```
df.write.format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/mi_base") \
    .option("dbtable", "usuarios") \
    .option("user", "postgres") \
    .option("password", "1234") \
    .save()
```

Ilustración 9 Escritura en una base de datos

ACTIVIDAD PRÁCTICA GUIADA: Análisis de datos con Spark SQL y DataFrames

Objetivo: Implementar Spark SQL para procesar datos estructurados, usando DataFrames, consultas SQL, UDFs y lectura/escritura en distintos formatos.

Requisitos previos

1. Tener **Apache Spark** instalado en el sistema.
2. Instalar **PySpark** en un entorno de Python (pip install pyspark).
3. Contar con un archivo de datos en formato **JSON o CSV**.

Para esta actividad, usaremos un dataset ficticio de transacciones bancarias llamado "transacciones.json" con la siguiente estructura:

```
[
  {"id": 1, "cliente": "Ana", "monto": 150.75, "categoria": "Compras"},
  {"id": 2, "cliente": "Carlos", "monto": 230.50, "categoria": "Viajes"},
  {"id": 3, "cliente": "Elena", "monto": 85.30, "categoria": "Alimentación"},
  {"id": 4, "cliente": "Miguel", "monto": 320.00, "categoria": "Electrónica"}
]
```

Paso 1: Configurar una sesión de Spark

Primero, creamos una **sesión Spark** que servirá como punto de entrada a Spark SQL.

```
from pyspark.sql import SparkSession

# Crear sesión Spark
spark = SparkSession.builder \
    .appName("AnálisisTransacciones") \
    .getOrCreate()
```


Paso 2: Cargar los datos en un DataFrame

Ahora, cargamos el archivo transacciones.json en un DataFrame de Spark.

```
df = spark.read.json("transacciones.json")
df.show()
```

Paso 3: Explorar el esquema del DataFrame

Es importante revisar el esquema del DataFrame para entender los tipos de datos.

```
df.printSchema()
```

Paso 4: Registrar el DataFrame como una tabla SQL

Podemos ejecutar consultas SQL registrando el DataFrame como una tabla temporal.

```
df.createOrReplaceTempView("transacciones")
consulta = spark.sql("SELECT cliente, monto FROM transacciones WHERE monto > 100")
consulta.show()
```

Paso 5: Crear una Función Definida por el Usuario (UDF)

Vamos a definir una **UDF** para clasificar las transacciones según su monto.

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def clasificar_monto(monto):
    return "Alta" if monto > 200 else "Baja"

# Convertimos la función en UDF
udf_clasificar = udf(clasificar_monto, StringType())

# Aplicamos la UDF
df = df.withColumn("categoria_monto", udf_clasificar(df["monto"]))
df.show()
```

Paso 6: Guardar el DataFrame en formato Parquet

Ahora, exportamos los datos transformados a **Parquet**, un formato eficiente para análisis de datos.

```
df.write.parquet("transacciones_procesadas.parquet")
```

Paso 7: Leer datos desde una base de datos (opcional)

Si queremos leer datos desde MySQL:

```
df_mysql = spark.read.format("jdbc") \
    .option("url", "jdbc:mysql://localhost:3306/mi_base") \
    .option("dbtable", "transacciones") \
    .option("user", "root") \
    .option("password", "1234") \
    .load()
df_mysql.show()
```