Ejercicio 2

- Objetivo: Procesar y analizar logs de servidor utilizando User-Defined Functions (UDFs) y operaciones de texto en Spark.
Planteamiento: Un equipo de operaciones de TI necesita analizar los logs de sus servidores para identificar patrones y problemas. Los logs contienen información como timestamp, nivel de log (INFO, ERROR, WARN), y mensaje.
- Crea un DataFrame simulando logs de servidor.
- Implementa una UDF para extraer el nivel de log de cada entrada.
- Utiliza funciones de procesamiento de texto para extraer el timestamp y el mensaje.
- Agrupa los logs por nivel y calcula la frecuencia de cada tipo.
- Ayuda:
  - Usa spark.udf.register() para crear una UDF que extraiga el nivel de log.
  - Las funciones substring() y regexp_extract() pueden ser útiles para procesar el texto de los logs.
  - Considera usar withColumn() para añadir nuevas columnas con la información extraída.
  - groupBy() y count() te ayudarán a calcular las frecuencias de los niveles de log.

```
import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.functions._


object Ejercicio2 {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder()

      .appName("Ejercicio2")

      .master("local[*]")

      .getOrCreate()


    import spark.implicits._


    // Simular logs
```

```
  val logs = Seq(
    "2023-05-01 10:00:15 INFO [User:123] Login successful",
    "2023-05-01 10:05:20 ERROR [User:456] Failed login attempt",
    "2023-05-01 10:10:30 WARN [System] High CPU usage detected",
    "2023-05-02 11:15:45 INFO [ServiceA] Request processed",
    "2023-05-02 11:20:50 ERROR [Database] Connection timed out",
    "2023-05-02 11:25:05 INFO [User:789] Logout successful",
    "2023-05-03 09:00:00 DEBUG [System] Starting background task",
    "2023-05-03 09:05:10 WARN [Network] Retrying connection"
  ).toDF("log")

  // UDF para extraer nivel de log
  val extractLevel = udf((log: String) => {
    val levels = Seq("INFO", "ERROR", "WARN", "DEBUG")
    levels.find(log.contains).getOrElse("UNKNOWN")
  })

  // Procesar logs
  val processedLogs = logs
    .withColumn("timestamp", to_timestamp(substring($"log", 1, 19), "yyyy-MM-dd
HH:mm:ss"))
    .withColumn("level", extractLevel($"log"))
    .withColumn("user_info", regexp_extract($"log", "\\[(.*?)\\]", 1))
    .withColumn("message", regexp_extract($"log", "\\] (.+)$", 1))
    .drop("log") // Eliminar la columna original de logs
```

```
println("Processed Logs:")
processedLogs.show(false)

// Calcular frecuencia de niveles de log
val logFrequency = processedLogs.groupBy("level").count()
println("\nLog Level Frequency:")
logFrequency.show()

// Calcular frecuencia de mensajes (ejemplo)
val messageFrequency =
processedLogs.groupBy("message").count().orderBy(desc("count"))
println("\nMessage Frequency:")
messageFrequency.show(truncate = false)

// Ejemplo de filtrado por nivel
val errorLogs = processedLogs.filter($"level" === "ERROR")
println("\nError Logs:")
errorLogs.show(false)

// Ejemplo de análisis por día
val logsByDay = processedLogs
  .withColumn("date", to_date($"timestamp"))
  .groupBy("date", "level")
  .count()
  .orderBy("date", "level")
println("\nLog Frequency by Day and Level:")
```

logsByDay.show()

spark.stop()

}

}

Como ejecutarlo:

- **Guardar:** Guarde el código como Ejercicio2.scala en el directorio de su proyecto Scala.
- **Configuración de Spark:** asegúrese de tener Apache Spark configurado en su entorno.
- **Compilación (si es necesario):** si usa una herramienta de compilación como SBT o Maven, asegúrese de que build.sbt o pom.xml incluya la dependencia de Spark SQL.
- **Ejecutar:** puede ejecutar este código mediante el comando spark-submit desde el directorio de instalación de Spark: Bashspark-submit --class Ejercicio2 your_project.jar

(Reemplace your_project.jar por el nombre real del archivo JAR compilado).
Al ejecutar este código, verá el siguiente resultado en la consola:

```
Processed Logs:
+-------------------+-----+---------+---------------------------+
|timestamp          |level|user_info|message                    |
+-------------------+-----+---------+---------------------------+
|2023-05-01 10:00:15|INFO |User:123 |Login successful           |
|2023-05-01 10:05:20|ERROR|User:456 |Failed login attempt       |
|2023-05-01 10:10:30|WARN |System   |High CPU usage detected    |
|2023-05-02 11:15:45|INFO |ServiceA |Request processed          |
|2023-05-02 11:20:50|ERROR|Database |Connection timed out       |
|2023-05-02 11:25:05|INFO |User:789 |Logout successful          |
|2023-05-03 09:00:00|DEBUG|System   |Starting background task   |
|2023-05-03 09:05:10|WARN |Network  |Retrying connection        |
+-------------------+-----+---------+---------------------------+

Log Level Frequency:
+-----+-----+
|level|count|
+-----+-----+
|WARN |2    |
|ERROR|2    |
|INFO |3    |
|DEBUG|1    |
+-----+-----+

Message Frequency:
+---------------------------+-----+
|message                    |count|
+---------------------------+-----+
|Login successful           |1    |
|Failed login attempt       |1    |
|High CPU usage detected    |1    |
|Request processed          |1    |
|Connection timed out       |1    |
|Logout successful          |1    |
```

```
|Starting background task    |1    |
|Retrying connection         |1    |
+----------------------------+-----+

Error Logs:
+-------------------+-----+--------+--------------------+
|timestamp          |level|userInfo|message             |
+-------------------+-----+--------+--------------------+
|2023-05-01 10:05:20|ERROR|User:456|Failed login attempt|
|2023-05-02 11:20:50|ERROR|Database|Connection timed out|
+-------------------+-----+--------+--------------------+

Log Frequency by Day and Level:
+----------+-----+-----+
|date      |level|count|
+----------+-----+-----+
|2023-05-01|ERROR|1    |
|2023-05-01|INFO |1    |
|2023-05-01|WARN |1    |
|2023-05-02|ERROR|1    |
|2023-05-02|INFO |2    |
|2023-05-03|DEBUG|1    |
|2023-05-03|WARN |1    |
+----------+-----+-----+
```

Explicación del Código Original:

Este programa en Scala utiliza Apache Spark para procesar una serie de cadenas de texto que simulan registros (logs) de un sistema.

1. **Importaciones:**
   - `org.apache.spark.sql.SparkSession`: Necesario para crear una sesión de Spark, que es el punto de entrada para trabajar con Spark SQL.
   - `org.apache.spark.sql.functions._`: Importa funciones útiles de Spark SQL, como `substring`, `to_timestamp`, `regexp_extract`, `udf`, etc.

2. **Objeto `Ejercicio2`:**
   - Define un objeto llamado `Ejercicio2`, que contiene el método `main`, el punto de entrada de la aplicación Scala.

3. **Creación de la Sesión de Spark:**
   - `val spark = SparkSession.builder().appName("Ejercicio2").master("local[*]").getOrCreate()`: Crea o obtiene una sesión de Spark.
     - `.builder()`: Inicia la construcción de la sesión.
     - `.appName("Ejercicio2")`: Asigna un nombre a la aplicación Spark.
     - `.master("local[*]")`: Indica que Spark se ejecute en modo local, utilizando todos los núcleos disponibles de la máquina.

- `.getOrCreate()`: Devuelve la sesión de Spark existente si ya hay una, o crea una nueva si no.

4. **Importación de Implícitos:**
   - `import spark.implicits._`: Importa implicaciones que permiten convertir secuencias de Scala (como la lista de logs) en DataFrames de Spark de forma más sencilla.

5. **Simulación de Logs:**
   - `val logs = Seq(...) .toDF("log")`: Crea una secuencia de cadenas de texto que representan líneas de log. Luego, `.toDF("log")` convierte esta secuencia en un DataFrame de Spark con una única columna llamada "log".

6. **Función Definida por el Usuario (UDF) `extractLevel`:**
   - `val extractLevel = udf((log: String) => { ... })`: Define una función que toma una cadena de log como entrada y devuelve el nivel del log ("INFO", "ERROR", "WARN" o "UNKNOWN" si no se encuentra ninguno).
     - `val levels = Seq("INFO", "ERROR", "WARN")`: Define una lista de niveles de log conocidos.
     - `levels.find(log.contains).getOrElse("UNKNOWN")`: Busca si alguna de las cadenas en `levels` está contenida en la cadena `log`. Si se encuentra, devuelve ese nivel; de lo contrario, devuelve "UNKNOWN".
     - `udf(...)`: Envuelve la función Scala en una UDF de Spark para que pueda ser utilizada en las operaciones del DataFrame.

7. **Procesamiento de Logs:**
   - `val processedLogs = logs ...`: Crea un nuevo DataFrame llamado `processedLogs` a partir del DataFrame `logs` original, aplicando varias transformaciones:
     - `.withColumn("timestamp", to_timestamp(substring($"log", 1, 19), "yyyy-MM-dd HH:mm:ss"))`: Crea una nueva columna llamada "timestamp" extrayendo los primeros 19 caracteres de la columna "log" (que se espera que contengan la marca de tiempo) y convirtiéndolos al tipo de dato `Timestamp` utilizando el formato especificado.
     - `.withColumn("level", extractLevel($"log"))`: Crea una nueva columna llamada "level" aplicando la UDF `extractLevel` a la columna "log".
     - `.withColumn("user_info", regexp_extract($"log", "\\[(.*?)\\]", 1))`: utiliza la función `withColumn` para añadir o reemplazar una columna

llamada `"user_info"` en un DataFrame. Vamos a desglosar cada parte en español:

- **`.withColumn("user_info", ...)`**: Esta es una función de transformación de DataFrames en Spark. Toma dos argumentos:
  - El primer argumento `"user_info"` es el nombre de la nueva columna que quieres crear o el nombre de una columna existente que quieres reemplazar.
  - El segundo argumento es la expresión que define los valores para esta nueva columna.
- **`regexp_extract(...)`**: Esta es una función de Spark SQL que se utiliza para extraer un grupo específico de una columna de tipo cadena basándose en una expresión regular. Toma tres argumentos:

  `$"log"`: Esto se refiere a una columna llamada `"log"` en tu DataFrame. El símbolo `$` es una notación abreviada en Spark SQL para referirse a una columna.

  `"\\[(.*?)\\]"`: Esta es la expresión regular:

  `\\[`: Coincide con un corchete de apertura literal `[`. La barra invertida `\` se utiliza para escapar el significado especial de `[`.

  `( ... )`: Esto crea un grupo de captura. La parte de la cadena que coincida con el patrón dentro de estos paréntesis será extraída.

  `.*?`: Esto coincide con cualquier carácter (`.`) cero o más veces (`*?`) de forma no codiciosa. No codicioso significa que coincidirá con la cadena más corta posible que satisfaga el patrón.

  `\\]`: Coincide con un corchete de cierre literal `]`. De nuevo, la barra invertida `\` escapa el significado especial de `]`.

  `1`: Este es el índice del grupo de captura que quieres extraer. Dado que solo tenemos un grupo de captura `(.*?)`, su índice es `1`. Si hubiera varios grupos de captura, podrías especificar un índice diferente para extraer un grupo distinto.

- `.withColumn("message", regexp_extract($"log", "\\] (.+)$", 1))`: Crea una nueva columna llamada "message" extrayendo la parte del mensaje de log que sigue al último corchete cerrado (`]`). La expresión regular `\\] (.+)$` busca un corchete cerrado seguido de un espacio y luego captura todo lo que viene hasta el final de la línea (el grupo de captura 1).

8. **Mostrar Logs Procesados:**
   - `processedLogs.show(false)`: Muestra el contenido del DataFrame `processedLogs`. `false` indica que no se trunquen las columnas largas.

9. **Calcular Frecuencia de Niveles de Log:**
    - `val logFrequency = processedLogs.groupBy("level").count()`: Agrupa el DataFrame `processedLogs` por la columna "level" y luego cuenta el número de filas en cada grupo, creando un nuevo DataFrame con la frecuencia de cada nivel.
    - `logFrequency.show()`: Muestra el contenido del DataFrame `logFrequency`.

10. **Detener la Sesión de Spark:**
    - `spark.stop()`: Libera los recursos utilizados por la sesión de Spark.

Explicación de las Modificaciones Añadidas:

En la versión modificada del código, he añadido algunas operaciones y salidas para demostrar cómo se puede seguir trabajando con el DataFrame procesado:

1. **Extracción de Información del Usuario/Sistema:**
    - `.withColumn("user_info", regexp_extract($"log", "\\[(.*?)\\]", 1))`: Se agrega una nueva columna llamada `user_info`. Utiliza la función `regexp_extract` con la expresión regular `\\[(.*?)\\]` para extraer el texto que se encuentra dentro de los corchetes `[]` en la columna "log". El `(.*?)` es un grupo de captura que coincide con cualquier carácter (`.`) cero o más veces (`*`), de forma no codiciosa (`?`). Esto permite extraer información como "User:123" o "System".

2. **Eliminación de la Columna Original "log":**
    - `.drop("log")`: Después de extraer la información relevante en columnas separadas, la columna original "log" se elimina utilizando `.drop("log")`. Esto puede hacer que el DataFrame sea más manejable y enfocado en los datos extraídos.

3. **Salida Adicional para Demostración:**
    - Se han añadido líneas `println(...)` antes de cada `show()` para proporcionar etiquetas descriptivas a la salida.
    - `processedLogs.first()`: Muestra la primera fila del DataFrame `processedLogs`. Esto puede ser útil para inspeccionar cómo se ven los datos después de las transformaciones.
    - `processedLogs.printSchema()`: Imprime el esquema del DataFrame `processedLogs`. El esquema muestra los nombres de las columnas y sus tipos de datos.

    - **Ejemplo de Filtrado por Nivel:**
        - `val errorLogs = processedLogs.filter($"level" === "ERROR")`: Crea un nuevo DataFrame llamado `errorLogs` que contiene solo las filas donde la columna "level" es igual a "ERROR".

        - `errorLogs.show(false)`: Muestra el contenido del DataFrame `errorLogs`.

    - **Ejemplo de Selección de Columnas Específicas:**
        - `val timestampsAndMessages = processedLogs.select("timestamp", "message")`: Crea un nuevo DataFrame llamado `timestampsAndMessages` que contiene solo las columnas "timestamp" y "message" del DataFrame `processedLogs`.
        - `timestampsAndMessages.show(false)`: Muestra el contenido del DataFrame `timestampsAndMessages`.

En resumen, el código toma datos de log simulados, los procesa para extraer información relevante como la marca de tiempo, el nivel del log y el mensaje, calcula la frecuencia de los diferentes niveles de log y, en la versión modificada, también extrae información adicional (como el usuario o el sistema), elimina la columna original de logs y demuestra algunas operaciones básicas como ver la primera fila, el esquema, filtrar datos y seleccionar columnas específicas.