Ejercicio 4

- Objetivo: Realizar operaciones de ETL (Extract, Transform, Load) en datos almacenados en formato Parquet y optimizar consultas.
- Planteamiento: Un equipo de análisis de datos necesita preparar un gran conjunto de datos de ventas para su análisis. Los datos están almacenados en formato Parquet y necesitan ser procesados, transformados y optimizados para consultas eficientes.
- Lee un conjunto de datos de ventas desde archivos Parquet.
- Realiza transformaciones como:
  - Convertir fechas a un formato estándar.
  - Calcular el total de ventas por transacción.
- Particiona los datos por fecha para optimizar las consultas.
- Escribe los datos transformados de vuelta en formato Parquet.
- Realiza y optimiza una consulta para obtener las ventas totales por mes.
- Ayuda:
  - Usa spark.read.parquet() para leer los datos Parquet.
  - Las funciones to_date() y date_format() pueden ser útiles para el manejo de fechas.
  - Considera usar partitionBy() al escribir los datos para optimizar futuras consultas.
  - Utiliza explain() para ver el plan de ejecución de tu consulta y buscar oportunidades de optimización.
- Dataset: "<https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-01.parquet>"

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

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

import java.net.URL

import java.nio.channels.Channels

import java.nio.file.{Files, Paths, StandardCopyOption}


object Ejercicio4 {

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

  val spark = SparkSession.builder()

   .appName("Ejercicio4")

   .master("local[*]")

   .config("spark.sql.parquet.compression.codec", "snappy")

   .getOrCreate()


  import spark.implicits._
```

```
// URL del archivo Parquet
val url = "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-01.parquet"

// Descargar el archivo
val file = Paths.get("yellow_tripdata_2023-01.parquet")
if (!Files.exists(file)) {
  println("Descargando archivo...")
  val connection = new URL(url).openConnection()
  val in = connection.getInputStream
  Files.copy(in, file, StandardCopyOption.REPLACE_EXISTING)
  in.close()
  println("Archivo descargado.")
}

// Leer el archivo Parquet
val trips = spark.read.parquet(file.toString)

// Realizar transformaciones
val processedTrips = trips
  .withColumn("trip_date", to_date($"tpep_pickup_datetime"))
  .withColumn("total_amount", $"total_amount".cast("double"))
  .withColumn("trip_duration",
    (unix_timestamp($"tpep_dropoff_datetime") - unix_timestamp($"tpep_pickup_datetime")) / 60)

// Particionar los datos por fecha para optimizar las consultas
processedTrips.write
  .partitionBy("trip_date")
  .mode("overwrite")
  .parquet("processed_trips")
```

```
    // Realizar y optimizar una consulta para obtener las ventas totales por mes
    val monthlyStats = spark.read.parquet("processed_trips")
      .groupBy(month($"trip_date").as("month"))
      .agg(
        sum("total_amount").as("total_sales"),
        avg("trip_duration").as("avg_duration"),
        count("*").as("trip_count")
      )
      .orderBy("month")

    // Mostrar el plan de ejecución
    println("Plan de Ejecución:")
    monthlyStats.explain()

    // Mostrar resultados
    println("\nVentas Totales, Duración Promedio y Conteo de Viajes por Mes (Enero de 2023):")
    monthlyStats.show()

    spark.stop()
  }
}
```

Explicación del Código:

Este programa en Scala utiliza Apache Spark para descargar, procesar y analizar un archivo Parquet que contiene datos de viajes en taxi amarillo de la ciudad de Nueva York para el mes de enero de 2023.

1. **Importaciones:** Se importan las clases necesarias de Spark SQL, funciones de Spark SQL, y clases de Java para trabajar con URLs y archivos.

2. **Creación de la Sesión de Spark:** Se crea una sesión de Spark con un nombre de aplicación y se configura para ejecutarse localmente. También se establece el códec de compresión para archivos Parquet a "snappy".

3. **URL del Archivo Parquet:** Se define la URL del archivo Parquet que se va a descargar.

4. **Descarga del Archivo:**
   - Se construye un objeto `Path` para el archivo local.
   - Se verifica si el archivo ya existe. Si no existe:
     - Se imprime un mensaje indicando que se está descargando el archivo.
     - Se abre una conexión a la URL.
     - Se obtiene un `InputStream` desde la conexión.
     - Se copia el contenido del `InputStream` al archivo local, reemplazando el archivo si ya existiera.
     - Se cierra el `InputStream`.
     - Se imprime un mensaje indicando que el archivo se ha descargado.

5. **Lectura del Archivo Parquet:** Se utiliza `spark.read.parquet()` para leer el archivo Parquet descargado en un DataFrame llamado `trips`.

6. **Realización de Transformaciones:**
   - `.withColumn("trip_date", to_date($"tpep_pickup_datetime"))`: Se crea una nueva columna llamada `trip_date` extrayendo la fecha de la columna `tpep_pickup_datetime`.
   - `.withColumn("total_amount", $"total_amount".cast("double"))`: Se asegura de que la columna `total_amount` tenga el tipo de dato `double`.
   - `.withColumn("trip_duration", ...)`: Se calcula la duración del viaje en minutos restando el timestamp de recogida del timestamp de entrega y dividiendo el resultado por 60.

7. **Particionamiento de los Datos:**
   - `processedTrips.write.partitionBy("trip_date").mode("overwrite").parquet("processed_trips")`: Se escribe el DataFrame `processedTrips` en formato Parquet en un directorio llamado `processed_trips`. Los datos se particionan por la columna `trip_date`. El particionamiento ayuda a optimizar las consultas posteriores que filtran por fecha, ya que Spark solo necesitará leer las particiones relevantes. El modo "overwrite" indica que, si el directorio ya existe, su contenido será reemplazado.

8. **Consulta para Obtener Estadísticas Mensuales:**

- `spark.read.parquet("processed_trips")`: Se lee el directorio particionado `processed_trips` en un nuevo DataFrame.
- `.groupBy(month($"trip_date").as("month"))`: Se agrupan las filas por el mes extraído de la columna `trip_date`. La función `month()` devuelve el número del mes (1 para enero, 2 para febrero, etc.), y se le asigna el alias "month".
- `.agg(...)`: Se realizan funciones de agregación para cada grupo (cada mes):
  - `sum("total_amount").as("total_sales")`: Calcula la suma de la columna `total_amount` y la renombra como `total_sales`.
  - `avg("trip_duration").as("avg_duration")`: Calcula el promedio de la columna `trip_duration` y la renombra como `avg_duration`.
  - `count("*").as("trip_count")`: Cuenta el número de filas en cada grupo (el número total de viajes por mes) y la renombra como `trip_count`.
- `.orderBy("month")`: Se ordenan los resultados por la columna "month".

9. **Mostrar el Plan de Ejecución:**
   - `monthlyStats.explain()`: Imprime el plan de ejecución lógico y físico de la consulta `monthlyStats`. Esto es útil para entender cómo Spark optimizará y ejecutará la consulta.

10. **Mostrar Resultados:**
    - `monthlyStats.show()`: Muestra el DataFrame `monthlyStats` en la consola, conteniendo las ventas totales, la duración promedio de los viajes y el número total de viajes para cada mes (en este caso, solo enero de 2023).

11. **Detener la Sesión de Spark:** Se cierra la sesión de Spark para liberar los recursos.

Posible Resultado en la Consola:

Primero, verás mensajes indicando la descarga del archivo si aún no existe. Luego, se imprimirá el plan de ejecución (que será extenso y mostrará las etapas de la consulta). Finalmente, se mostrarán los resultados de la consulta `monthlyStats`:

```
Plan de Ejecución:
... (aquí se mostrará el plan de ejecución de Spark) ...

Ventas Totales, Duración Promedio y Conteo de Viajes por Mes (Enero de 2023):
+-----+-----------+------------+----------+
|month|total_sales|avg_duration|trip_count|
+-----+-----------+------------+----------+
|1    |XXXXXXXXX.XX|YYYYY.YYY   |ZZZZZZZZ  |
+-----+-----------+------------+----------+
```

Nota:

- `XXXXXXXXX.XX` será la suma total de la columna `total_amount` para todos los viajes en enero de 2023.
- `YYYYY.YYY` será el promedio de la columna `trip_duration` (en minutos) para todos los viajes en enero de 2023.
- `ZZZZZZZZ` será el número total de filas (viajes) en el archivo de enero de 2023.

Dado que solo estamos procesando los datos de enero de 2023, la tabla de resultados solo tendrá una fila correspondiente al mes 1. Los valores exactos dependerán del contenido del archivo Parquet descargado.