```
import org.apache.spark.sql.{SparkSession, DataFrame}
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.storage.StorageLevel
import java.net.URL
import java.io.{File, FileOutputStream}

object CompleteExample2 {
  def main(args: Array[String]): Unit = {
    // Inicializar la sesión de Spark con configuraciones optimizadas
    val spark = SparkSession.builder()
      .appName("CompleteExample2")
      .master("local[*]")
      .config("spark.sql.warehouse.dir", "spark-warehouse")
      .config("spark.executor.memory", "2g")
      .config("spark.driver.memory", "2g")
      .config("spark.sql.shuffle.partitions", "8")
      .getOrCreate()

    import spark.implicits._

    // URL del conjunto de datos de vuelos (sustituye con una URL real si esta no funciona)
    val url = "https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat"

    // Descargar el archivo CSV
    val localFilePath = "routes.csv"
    downloadFile(url, localFilePath)
    println(s"Archivo CSV descargado en: $localFilePath")

    // Leer el archivo CSV
    val dfRoutes = spark.read
```

```
  .option("header", "false") // El archivo no tiene encabezado
  .option("inferSchema", "true")
  .csv(localFilePath)
  .toDF("airline", "airline_id", "source_airport", "source_airport_id",
       "destination_airport", "destination_airport_id", "codeshare",
       "stops", "equipment")

// Mostrar el esquema y algunas filas de muestra
dfRoutes.printSchema()
dfRoutes.show(5)

// Persistir el DataFrame en memoria y disco para mejorar el rendimiento
dfRoutes.persist(StorageLevel.MEMORY_AND_DISK)

// Definir una UDF para categorizar las rutas basadas en el número de paradas
val categorizeRoute = udf((stops: Int) => {
  if (stops == 0) "Direct"
  else if (stops == 1) "One Stop"
  else "Multiple Stops"
})

// Aplicar la UDF y realizar algunas transformaciones
val dfProcessed = dfRoutes
  .withColumn("route_type", categorizeRoute($"stops"))
  .withColumn("is_international",
            when($"source_airport".substr(1, 2) =!= $"destination_airport".substr(1, 2), true)
            .otherwise(false))

// Crear una vista temporal para usar SQL
dfProcessed.createOrReplaceTempView("flight_routes")
```

```
// Realizar un análisis complejo usando SQL
val routeAnalysis = spark.sql("""
  SELECT
    airline,
    route_type,
    COUNT(*) as route_count,
    SUM(CASE WHEN is_international THEN 1 ELSE 0 END) as international_routes
  FROM flight_routes
  GROUP BY airline, route_type
  ORDER BY route_count DESC """)

println("Análisis de rutas por aerolínea:")
routeAnalysis.show()

// Utilizar funciones de ventana para análisis más avanzado
val windowSpec = Window.partitionBy("airline").orderBy($"route_count".desc)

val topRoutesByAirline = routeAnalysis
  .withColumn("rank", dense_rank().over(windowSpec))
  .filter($"rank" <= 3) // Obtener las top 3 tipos de rutas para cada aerolínea
  .orderBy($"airline", $"rank")

println("Top 3 tipos de rutas por aerolínea:")
topRoutesByAirline.show()

// Realizar un análisis de conectividad de aeropuertos
val airportConnectivity = dfProcessed
  .groupBy("source_airport")
  .agg(
    countDistinct("destination_airport").as("destinations"),
    sum(when($"is_international", 1).otherwise(0)).as("international_connections")
```

```
    )
    .orderBy($"destinations".desc)

  println("Análisis de conectividad de aeropuertos:")
  airportConnectivity.show()

  // Guardar los resultados en formato Parquet
  routeAnalysis.write.mode("overwrite").parquet("route_analysis.parquet")
  topRoutesByAirline.write.mode("overwrite").parquet("top_routes_by_airline.parquet")
  airportConnectivity.write.mode("overwrite").parquet("airport_connectivity.parquet")

  // Liberar el caché y detener la sesión de Spark
  dfRoutes.unpersist()
  spark.stop()
}

// Función auxiliar para descargar el archivo
def downloadFile(url: String, localFilePath: String): Unit = {
  val connection = new URL(url).openConnection()
  val inputStream = connection.getInputStream
  val outputStream = new FileOutputStream(new File(localFilePath))

  try {
    val buffer = new Array[Byte](4096)
    var bytesRead = inputStream.read(buffer)
    while (bytesRead != -1) {
      outputStream.write(buffer, 0, bytesRead)
      bytesRead = inputStream.read(buffer)
    }
  } finally {
    inputStream.close()
```

```
    outputStream.close()
  }
 }
}
```

Explicación del Código `CompleteExample2`

Este código Scala utiliza Apache Spark para analizar un conjunto de datos de rutas aéreas. El programa realiza las siguientes acciones:

1. **Inicialización de la Sesión de Spark:**
   - Crea o obtiene una sesión de Spark (`SparkSession`) con configuraciones similares al ejemplo anterior, optimizadas para la ejecución local.
2. **Descarga del Archivo CSV:**
   - Define la URL de un archivo CSV que contiene datos de rutas aéreas (este archivo proviene del proyecto OpenFlights). **Es importante tener en cuenta que la disponibilidad y el contenido exacto de este archivo pueden cambiar.**
   - Utiliza la misma función `downloadFile` del ejemplo anterior para descargar el archivo CSV desde la URL y guardarlo localmente como `routes.csv`.
3. **Lectura del Archivo CSV:**
   - Lee el archivo CSV descargado en un DataFrame llamado `dfRoutes`.
   - Se especifica que el archivo no tiene encabezado (`.option("header", "false")`).
   - Spark intenta inferir el esquema de los datos (`.option("inferSchema", "true")`).
   - Se asignan nombres explícitos a las columnas utilizando `.toDF()`. Estos nombres son: `airline`, `airline_id`, `source_airport`, `source_airport_id`, `destination_airport`, `destination_airport_id`, `codeshare`, `stops`, y `equipment`.
4. **Inspección del DataFrame:**
   - `dfRoutes.printSchema()` muestra el esquema del DataFrame.
   - `dfRoutes.show(5)` muestra las primeras 5 filas del DataFrame para dar una idea de los datos.
5. **Persistencia del DataFrame:**
   - `dfRoutes.persist(StorageLevel.MEMORY_AND_DISK)` persiste el DataFrame en memoria y disco para mejorar el rendimiento de las operaciones posteriores.
6. **Definición de una UDF (User Defined Function):**
   - Se define una UDF llamada `categorizeRoute` que toma el número de paradas (`stops`) como entrada (un entero) y devuelve una cadena que categoriza la ruta como "Direct" (0 paradas), "One Stop" (1 parada) o "Multiple Stops" (más de 1 parada).

7. **Transformaciones del DataFrame:**
    - Se crea un nuevo DataFrame llamado `dfProcessed` aplicando las siguientes transformaciones:
        - `withColumn("route_type", categorizeRoute($"stops"))`: Aplica la UDF `categorizeRoute` a la columna `stops` para crear una nueva columna llamada `route_type`.
        - `withColumn("is_international", ...)`: Crea una columna booleana llamada `is_international`. Se considera que una ruta es internacional si los dos primeros caracteres del código del aeropuerto de origen (`source_airport`) son diferentes de los dos primeros caracteres del código del aeropuerto de destino (`destination_airport`). Esto es una heurística simple y puede no ser completamente precisa para todos los códigos de aeropuerto.
8. **Creación de una Vista Temporal:**
    - `dfProcessed.createOrReplaceTempView("flight_routes")` crea una vista temporal llamada `flight_routes` basada en el DataFrame `dfProcessed`, permitiendo realizar consultas SQL.
9. **Análisis Complejo Usando SQL:**
    - Se ejecuta una consulta SQL contra la vista temporal `flight_routes` para realizar un análisis de rutas por aerolínea.
    - Selecciona la aerolínea (`airline`), el tipo de ruta (`route_type`), cuenta el número de rutas para cada combinación (`COUNT(*) as route_count`), y cuenta el número de rutas internacionales para cada combinación (`SUM(CASE WHEN is_international THEN 1 ELSE 0 END) as international_routes`).
    - Los resultados se agrupan por aerolínea y tipo de ruta, y se ordenan por el número de rutas en orden descendente.
    - Los resultados se almacenan en el DataFrame `routeAnalysis` y se muestran en la consola.
10. **Utilización de Funciones de Ventana:**
    - Se define una especificación de ventana (`windowSpec`) que particiona los datos por aerolínea (`airline`) y los ordena por el número de rutas (`route_count`) en orden descendente dentro de cada partición.
    - Se aplica la función de ventana `dense_rank()` sobre `windowSpec` para asignar un rango a cada tipo de ruta dentro de cada aerolínea, basado en el número de rutas. El resultado se guarda en una nueva columna llamada `rank`.
    - Se filtra el DataFrame resultante para mantener solo las filas donde el rango es menor o igual a 3, obteniendo así los 3 tipos de rutas más frecuentes para cada aerolínea.
    - Los resultados se ordenan por aerolínea y luego por rango, y se muestran en la consola.
11. **Análisis de Conectividad de Aeropuertos:**
    - Se realiza un análisis para determinar la conectividad de los aeropuertos.
    - Se agrupan los datos por el aeropuerto de origen (`source_airport`).
    - Se calcula el número de aeropuertos de destino distintos (`countDistinct("destination_airport").as("destinations")`) para cada aeropuerto de origen.

- Se calcula el número de conexiones internacionales (`sum(when($"is_international", 1).otherwise(0)).as("international_connections")`) para cada aeropuerto de origen.
- Los resultados se ordenan por el número de destinos en orden descendente y se muestran en la consola.

12. **Guardado de Resultados en Formato Parquet:**
    - Los DataFrames resultantes del análisis (`routeAnalysis`, `topRoutesByAirline`, y `airportConnectivity`) se guardan en formato Parquet en los directorios `route_analysis.parquet`, `top_routes_by_airline.parquet`, y `airport_connectivity.parquet`, respectivamente, sobrescribiendo cualquier archivo existente.
13. **Liberación del Caché y Detención de la Sesión de Spark:**
    - Se libera el DataFrame `dfRoutes` del caché y se detiene la sesión de Spark.

Posible Resultado del Código

El resultado del código se mostrará en la consola en tres partes principales:

1. Análisis de rutas por aerolínea:

Esta sección mostrará una tabla con las aerolíneas, los tipos de ruta (Direct, One Stop, Multiple Stops), el número total de rutas para esa combinación y el número de rutas internacionales para esa combinación. Los resultados estarán ordenados por el número de rutas en orden descendente. El resultado podría tener un formato similar a este (los datos exactos dependerán del archivo `routes.dat`):

```
Análisis de rutas por aerolínea:
+-------+-------------+-----------+----------------------+
|airline|   route_type|route_count|international_routes|
+-------+-------------+-----------+----------------------+
|    2B |Multiple Stops|       1500|                   900|
|    8V |     One Stop|       1200|                   750|
|    9W |     Multiple Stops|       1100|                   600|
|    ... |          ...|       ...|                   ...|
+-------+-------------+-----------+----------------------+
```

2. Top 3 tipos de rutas por aerolínea:

Esta sección mostrará las 3 categorías de rutas más frecuentes para cada aerolínea, ordenadas por aerolínea y luego por rango (basado en el número de rutas). El resultado podría tener un formato similar a este:

```
Top 3 tipos de rutas por aerolínea:
+-------+-------------+-----------+----------------------+----+
|airline|   route_type|route_count|international_routes|rank|
+-------+-------------+-----------+----------------------+----+
|    2B |Multiple Stops|       1500|                   900|   1|
|    2B |     One Stop|       1000|                   600|   2|
|    2B |       Direct|        500|                   300|   3|
```

```
|    8V |     One Stop|       1200|                    750|   1|
|    8V |Multiple Stops|        900|                     500|   2|
|    8V |       Direct|        700|                    400|   3|
|    9W |Multiple Stops|       1100|                     600|   1|
|    9W |     One Stop|        800|                    450|   2|
|    9W |       Direct|        600|                    350|   3|
|    ... |          ...|       ...|                    ...| ...|
+-------+-------------+-----------+-----------------------+----+
```

3. Análisis de conectividad de aeropuertos:

Esta sección mostrará los aeropuertos de origen, el número de destinos distintos a los que tienen rutas y el número de conexiones internacionales que tienen, ordenados por el número de destinos en orden descendente. El resultado podría tener un formato similar a este (los códigos de aeropuerto exactos dependerán del archivo `routes.dat`):

```
Análisis de conectividad de aeropuertos:
+--------------+------------+--------------------------+
|source_airport|destinations|international_connections|
+--------------+------------+--------------------------+
|           ATL|         300|                       200|
|           ORD|         280|                       180|
|           LHR|         250|                       220|
|           ...|         ...|                       ...|
+--------------+------------+--------------------------+
```

Además de la salida en la consola, el programa creará tres directorios en el sistema de archivos local:

- `route_analysis.parquet`
- `top_routes_by_airline.parquet`
- `airport_connectivity.parquet`

Estos directorios contendrán los DataFrames resultantes guardados en formato Parquet.