**ASSIGNMENT DATE-26-12-2023**

PySpark, the Python API for Apache Spark, has revolutionized the way we handle Big Data. It’s an ETL powerhouse that combines the simplicity of Python with the scalability and performance of Spark.

Why Choose PySpark for ETL?  
🔹Performance: PySpark leverages in-memory computing, making ETL processes faster than ever.  
🔹Ease of Use: Python developers can seamlessly transition to PySpark due to its Pythonic syntax.  
🔹Scalability: Handle massive datasets with ease, thanks to Spark’s distributed processing.  
🔹Rich Ecosystem: PySpark integrates with popular tools and libraries, making it versatile for various data tasks.

The PySpark ETL Workflow:  
🔹Extract: Retrieve data from various sources like databases, files, or APIs.  
🔹Transform: Clean, aggregate, and manipulate data to fit your analysis needs.  
🔹Load: Store the transformed data into a database or data warehouse for analysis.

Real-World Applications:  
Perform large-scale data cleansing and preparation.  
Analyze streaming data in real-time.  
Create data pipelines for machine learning and AI applications.  
Handle structured and unstructured data effortlessly.

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, concat, lit, floor, rand

# Initialize a Spark session  
spark = SparkSession.builder.appName(“ComplexETL”).getOrCreate()

# Define the external source and target paths  
source\_path = ‘your\_path’ # Update with your actual source file path  
target\_path = “your\_output\_path” # Update with your desired target file path

# Extract: Read data from an external CSV file  
df = spark.read.csv(source\_path, header=True,schema = ‘cust\_id int, first\_name string,last\_name string,cust\_order int,cust\_status string’)

# Transformation 1: Concatenate First and Last Names  
df = df.withColumn(“full\_name”, concat(col(“first\_name”), lit(“ “), col(“last\_name”)))

# Transformation 2: Calculate Net Salary (subtract 10% as taxes)  
df = df.withColumn(“net\_salary”, floor(lit(10000) + rand() \* lit(50)) )  
#adding age column  
df = df.withColumn(“age”, floor(lit(20) + rand() \* lit(31)))

# Transformation 3: Filter by Age (age >= 30)  
df = df.filter(col(“age”) >= 30)

# Transformation 4: Group by Age and Calculate Average Salary  
avg\_salary\_by\_age = df.groupBy(“age”).agg({“net\_salary”: “avg”}).withColumnRenamed(“avg(salary)”, “avg\_salary”)

# Transformation 5: Sort by Age  
df = df.orderBy(“age”)

# Save the transformed data to an external CSV file  
df.write.csv(target\_path, mode=”overwrite”, header=True)