Raport

Lab. 4

**Autor Maciej Tonderski**

1. **Cel ćwieczenia**

Celem tego ćwiczenia jest przeprowadzenie analizy danych transakcyjnych detalicznych za pomocą Apache Spark SQL. Studenci będą pracować na zbiorze danych "Online Retail", który zawiera informacje o transakcjach detalicznych, takie jak identyfikator transakcji, kod produktu, ilość, cena jednostkowa, kraj itp. Zadanie będzie polegać na wczytaniu danych, wykonaniu różnych analiz, takich jak analiza sprzedaży, trendów produktowych, analiza krajów itp., oraz prezentacji wyników za pomocą odpowiednich zapytań SQL i wizualizacji.

Do ćwiczenia stworzono skrypt:  
import org.apache.spark.sql.SparkSession

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

import org.apache.spark.sql.expressions.Window

val spark = SparkSession.builder.appName("Online Retail Analysis").getOrCreate()

println("Step 1: Reading the Data")

// Step 1: Reading the Data

val df = spark.read.option("header", "true").option("inferSchema", "true").csv("/opt/spark-data/OnlineRetail.csv")

println("Displaying the first 20 rows of the dataset:")

df.show()

println("Step 2: Data Cleaning")

// Step 2: Data Cleaning

println("Checking for missing values and dropping rows with missing values")

val cleanedDF = df.na.drop()

println("Checking for duplicates and dropping duplicate rows")

val deduplicatedDF = cleanedDF.dropDuplicates()

println("Displaying the cleaned data:")

deduplicatedDF.show()

println("Step 3: Sales Analysis")

// Step 3: Sales Analysis

println("Calculating the total sales value for each transaction")

val salesDF = deduplicatedDF.withColumn("TotalSale", col("Quantity") \* col("UnitPrice"))

val totalSalesDF = salesDF.groupBy("InvoiceNo").agg(sum("TotalSale").alias("TotalSale"))

println("Displaying the total sales per transaction, sorted by total sales:")

totalSalesDF.orderBy(desc("TotalSale")).show()

println("Step 4: Product Trends Analysis")

// Step 4: Product Trends Analysis

println("Analyzing product sales trends over time")

val windowSpec = Window.partitionBy("StockCode").orderBy("InvoiceDate")

val salesWithLagDF = salesDF.withColumn("PreviousTotalSale", lag("TotalSale", 1).over(windowSpec))

val salesTrendDF = salesWithLagDF.withColumn("PercentageChange",

(col("TotalSale") - col("PreviousTotalSale")) / col("PreviousTotalSale") \* 100)

println("Displaying the product sales trends with percentage changes:")

salesTrendDF.show()

println("Step 5: Geographical Analysis")

// Step 5: Geographical Analysis

println("Calculating total sales by country")

val countrySalesDF = salesDF.groupBy("Country").agg(sum("TotalSale").alias("TotalSale"))

println("Displaying the total sales per country, sorted by total sales:")

countrySalesDF.orderBy(desc("TotalSale")).show()

// Save results to CSV

println("Saving results to CSV files")

totalSalesDF.write.option("header", "true").csv("/opt/spark-data/output/total\_sales")

salesTrendDF.write.option("header", "true").csv("/opt/spark-data/output/sales\_trend")

countrySalesDF.write.option("header", "true").csv("/opt/spark-data/output/country\_sales")

println("Analysis complete. Results saved to /opt/spark-data/output/")

spark.stop()

Następnie w celu uruchomienia kodu stworzono dockerfile budujący obraz wraz z potrzebnymi bibliotekami:

# Use an official Spark base image

FROM bitnami/spark:latest

# Copy the Spark application code into the Docker image

COPY lab3.scala /opt/bitnami/spark/work-dir/

# Set the working directory

WORKDIR /opt/bitnami/spark/work-dir/

# Run the Spark Shell with the application code

CMD ["spark-shell", "-i", "lab3.scala"]

A na koniec do uruchomienia kodu stworzono docker-compose-yml file which creates master-slave spark system to execute code with attached volumes:

version: "3.7"

services:

spark-master:

image: bitnami/spark:latest

container\_name: spark-master

environment:

- SPARK\_MODE=master

ports:

- "8080:8080"

- "7077:7077"

spark-worker:

image: bitnami/spark:latest

container\_name: spark-worker

environment:

- SPARK\_MODE=worker

- SPARK\_MASTER\_URL=spark://spark-master:7077

depends\_on:

- spark-master

ports:

- "8081:8081"

spark-driver:

build: .

container\_name: spark-driver

environment:

- SPARK\_MASTER\_URL=spark://spark-master:7077

depends\_on:

- spark-master

- spark-worker

volumes:

- ./data:/opt/spark-data

To run it type in a console:

docker compose up --build

This action will build driver container and create rest of the needed containers alongside with respected networks.

1. **Wykonanie**

Zrealizowany skrypt wykonał kod którego poszególne kroki zostały opisane w załączeniu wklejony wynik z konsoli po uruchomieniu programu:

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

1. **Wnioski:**

Przeprowadzone ćwiczenie pokazało, jak efektywnie można wykorzystać Apache Spark SQL do analizy dużych zbiorów danych transakcyjnych. Dzięki zastosowaniu technik czyszczenia danych, grupowania i agregacji, udało się uzyskać wartościowe informacje, które mogą być użyteczne w podejmowaniu decyzji biznesowych. Analiza sprzedaży, trendów produktowych oraz analiza geograficzna dostarczyły istotnych wniosków, które mogą pomóc w optymalizacji strategii sprzedażowej i marketingowej. Dodatkowo, zastosowanie środowiska Docker pozwoliło na łatwe uruchomienie i skalowanie aplikacji w różnych środowiskach, co zwiększa elastyczność i efektywność przeprowadzanych analiz.