Assignment – Day 14

-Sarthak Niranjan Kulkarni (Maverick)

- [sarthakkul2311@gmail.com](mailto:sarthakkul2311@gmail.com) - (+91) 93256 02791

**21/11/2024 (Thursday)**

**Spark-SQL Practice: -**

1. **Loading a CSV file into a Spark DataFrame with specified options and creating a temporary SQL view.**

# File location and type

file\_location = "/FileStore/tables/simple\_zipcodes-1.csv"

file\_type = "csv"

# CSV options

infer\_schema = "false"

first\_row\_is\_header = "false"

delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type) \

  .option("inferSchema", infer\_schema) \

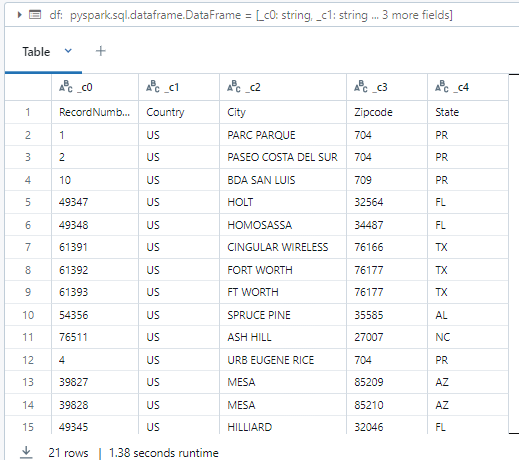
  .option("header", first\_row\_is\_header) \

  .option("sep", delimiter) \

  .load(file\_location)

display(df)

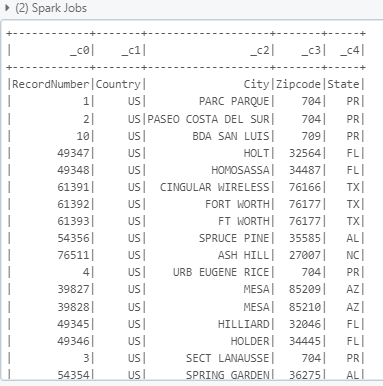
df.createOrReplaceTempView("tempdata")



1. **Querying a temporary SQL view and selecting specific columns from a DataFrame in Spark.**

spark.sql("select \* from tempdata").show()

df.select("\_c0","\_c1").show(5)

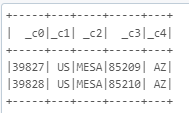
****

**A screenshot of a computer program

Description automatically generated**

1. **Filtering rows from a temporary SQL view in Spark based on a column value condition.**

spark.sql("""SELECT \* From tempdata WHERE \_c4='AZ'""").show(5)

****

1. **Loading a CSV file with headers into a Spark DataFrame and creating a temporary SQL view named "customer".**

# File location and type

file\_location = "/FileStore/tables/simple\_zipcodes-1.csv"

file\_type = "csv"

# CSV options

infer\_schema = "false"

first\_row\_is\_header = "true"

delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type) \

  .option("inferSchema", infer\_schema) \

  .option("header", first\_row\_is\_header) \

  .option("sep", delimiter) \

  .load(file\_location)

display(df)

df.createOrReplaceTempView("customer")

**A screenshot of a data

Description automatically generated**

1. **Displaying all records from a SQL view and selecting specific columns from a DataFrame in Spark.**

spark.sql("select \* from customer").show()

df.select("RecordNumber","Country").show(5)

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. **Filtering rows from the "customer" SQL view in Spark where the state is 'PR'.**

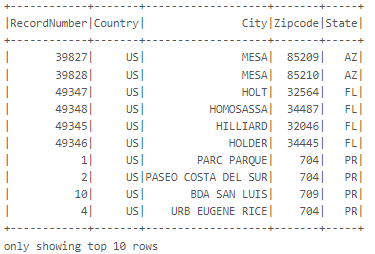
spark.sql("""SELECT \* From customer WHERE state='PR'""").show(5)

**A white paper with black text

Description automatically generated**

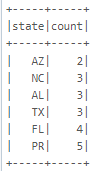
1. **Filtering and ordering rows from the "customer" SQL view in Spark by specific states.**

spark.sql("""select \* FROM customer WHERE state in ('PR','AZ','FL')order by state """).show(10)

****

1. **Grouping and counting the number of records for each state from the "customer" SQL view in Spark.**

spark.sql("""SELECT state,count(\*) as count FROM customer GROUP BY state""").show()

****

**Spark-SQL Summary: -**

The provided code demonstrates the process of loading and processing CSV files using Apache Spark. Initially, a CSV file is read into a Spark DataFrame with no schema inference and no header, then displayed and queried through a temporary SQL view named tempdata. Queries include displaying all data, selecting specific columns (\_c0, \_c1), and filtering rows where column \_c4 equals 'AZ'.

Later, the CSV file is reloaded with headers enabled into a DataFrame and assigned to a new SQL view named customer. Queries on this view include retrieving all data, selecting specific columns (RecordNumber, Country), filtering rows where the state column equals 'PR', and filtering states in a specific list (PR, AZ, FL) while ordering the results by state. Additionally, a query is used to group the data by the state column and count the number of records for each state. The code showcases Spark's SQL and DataFrame APIs for data analysis and transformations. The code also demonstrates how to read the same CSV file multiple times with different configurations, such as enabling header parsing in the second instance. Through these queries, Spark's ability to handle large datasets efficiently with SQL-like syntax for filtering, grouping, and ordering operations is highlighted, making it suitable for big data processing tasks.