Optimized Practical Codes for Spark

Practical 2: Filtering RDDs and Finding Minimum Temperature by Location

```
from pyspark import SparkConf, SparkContext

# Initialize Spark
conf = SparkConf().setMaster("local").setAppName("MinTemperature")
sc = SparkContext(conf=conf)

# Function to parse lines
def parseLine(line):
    fields = line.split(',')
    return fields[0], fields[2], float(fields[3]) * 0.1 * (9/5) + 32

# Load data and filter TMIN records
lines = sc.textFile("1800.csv")
minTemps = (
    lines.map(parseLine)
    .filter(lambda x: x[1] == "TMIN")
    .map(lambda x: (x[0], x[2]))
    .reduceByKey(min)
)

# Show results
for station, temp in minTemps.collect():
    print(f"{station}\t{temp:.2f}F")
```

Practical 3: Counting Word Occurrences using flatMap

```
from pyspark import SparkConf, SparkContext

# Initialize Spark
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)

# Load data and count words
lines = sc.textFile("text.txt")
wordCounts = (
```

```
    lines.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda x, y: x + y)
)

# Show results
for word, count in wordCounts.collect():
    print(f"{word}: {count}")
```

Practical 4: Executing SQL Commands on DataFrames

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder.appName("SQLCommands").getOrCreate()

# Load data into DataFrame
df = spark.read.csv("data.csv", header=True, inferSchema=True)

# Register DataFrame as SQL table
df.createOrReplaceTempView("data")

# Execute SQL queries
result = spark.sql("SELECT column1, column2 FROM data WHERE column3 > 100")

# Show results
result.show()
```

Practical 5: Total Spent by Customer with DataFrames

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder.appName("TotalSpent").getOrCreate()

# Load data into DataFrame
df = spark.read.csv("customer-orders.csv", header=False, inferSchema=True)
df = df.withColumnRenamed("_c0", "CustomerID").withColumnRenamed("_c2", "Amount")

# Calculate total spent
totalSpent =
```

```
df.groupBy("CustomerID").sum("Amount").withColumnRenamed("sum(Amount)",
"TotalSpent")

# Show results
totalSpent.show()
```

Practical 6: Use Broadcast Variables for Movie Names

```
from pyspark import SparkConf, SparkContext

# Initialize Spark
conf = SparkConf().setMaster("local").setAppName("BroadcastMovieNames")
sc = SparkContext(conf=conf)

# Load movie names into a dictionary
movieNames = {1: "Toy Story", 2: "Jumanji", 3: "Grumpier Old Men"}
broadcastMovieNames = sc.broadcast(movieNames)

# Load data and map movie IDs to names
lines = sc.textFile("ratings.csv")
movies = lines.map(lambda x: int(x.split(',')[1])).map(lambda id:
broadcastMovieNames.value.get(id, "Unknown"))

# Show results
for movie in movies.collect():
    print(movie)
```

Practical 7: Using Spark ML for Movie Recommendations

```
from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS

# Initialize SparkSession
spark = SparkSession.builder.appName("MovieRecommendation").getOrCreate()

# Load data into DataFrame
data = spark.read.csv("ratings.csv", header=True, inferSchema=True)

# Prepare ALS model
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId",
ratingCol="rating")
```

```
model = als.fit(data)

# Make recommendations for all users
recommendations = model.recommendForAllUsers(5)

# Show results
recommendations.show()
```

Practical 8: Structured Streaming with Most-Viewed URLs

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder.appName("StructuredStreaming").getOrCreate()

# Read streaming data
stream = spark.readStream.format("socket").option("host", "localhost").option("port",
9999).load()

# Process data
urls = stream.selectExpr("CAST(value AS STRING)").groupBy("value").count()

# Write streaming results to console
query = urls.writeStream.outputMode("complete").format("console").start()
query.awaitTermination()
```