Name:-Preksha Patel

Sapid:-60004210126

Branch:-Computer Engineering

Div:-C2 ; Batch:-1

BIG DATA INFRASTRUCTURE(BDI)

Experiment- 07

BDI

Name: Preksha A. Patel
Sapid: 60004210126
Branch: Computer Engineering
Div: C2, Batch: L

Ass: Experiment no. 07

Aim :- Implement RDD using Pyspark.

Theory :-

RDD stands for "Resilient Distributed Dataset".
It is the fundamental data structure of Apache Spark.
- RDD in Apache Spark is an immutable collection of objects which computes on different nodes of the cluster.
- Decomposing the RDD.

1. Resilient :- i.e. fault tolerant with the help of RDD linear graph (DAG) and so able to recompute missing or damaged partitions due to node failures.
2. Distributed :- Since data resides on multiple nodes.
3. Dataset represents data you work with. The user can load the dataset externally which can be either JSON file, CSV file, or database via JDBC with no specific data structure.

Features of RDD :-

- Resilience :- RDD's track data lineage information to recover lost data, automatically on failure. It is also called fault tolerance.
- Distributed :- Data present in RDD resides on multiple nodes. It is distributed across different nods of a cluster.
- In Memory computation :- An RDD stores any immediate data that is generated in the memory than on the disk so that it provides faster access.

- partitioning : partitions can be done on any existing RDD to create logical parts that are mutable you can achieve this by applying transformations in the existing partition.
- Immutability : Data is stored in an RDD in the read only mode you cannot edit the data within it present in RDD. But you can create new RD's by performing transformations in the existing RDD's.
- Lazy Evaluation : Data does not get loaded in RDD even if you define it transformations are actually computed when you call action.

Conclusion :
RDD is implemented successfully

1. **RDD for Armstrong Numbers between 100 and 9999.Code:**

```
from pyspark import SparkContext, SparkConf
import math

# Function to check if a number is Armstrong
def is_armstrong(num):
    num_str = str(num)
    num_digits = len(num_str)
    armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
    return armstrong_sum == num

# Create a SparkContext
conf = SparkConf().setAppName("ArmstrongNumbers").setMaster("local")
sc = SparkContext(conf=conf)

# Create an RDD containing numbers from 100 to 9999
numbers_rdd = sc.parallelize(range(100, 10000))

# Filter RDD to get Armstrong numbers
armstrong_rdd = numbers_rdd.filter(is_armstrong)

# Collect and print Armstrong numbers
armstrong_numbers = armstrong_rdd.collect()
for armstrong_number in armstrong_numbers:
    print(armstrong_number)

# Stop the SparkContext
sc.stop()
```

Output:

```
153
370
371
407
1634
8208
9474
```

2. **RDD for Perfect Numbers between 1 and 100 Code:**

```
Afrom pyspark import SparkContext, SparkConf
import math

# Create a SparkContext
conf = SparkConf().setAppName("PerfectNumbers").setMaster("local")
sc = SparkContext(conf=conf)

# Define a function to check if a number is perfect
def is_perfect(num):
    divisors_sum = sum([i for i in range(1, num) if num % i == 0])
    return divisors_sum == num

# Create an RDD containing numbers from 1 to 100
numbers_rdd = sc.parallelize(range(1, 101))

# Filter the RDD to get perfect numbers
perfect_numbers_rdd = numbers_rdd.filter(is_perfect)

# Collect and print perfect numbers
perfect_numbers = perfect_numbers_rdd.collect()
print("Perfect numbers between 1 and 100:")
for perfect_number in perfect_numbers:
    print(perfect_number)

# Stop the SparkContext
sc.stop()
```

Output:

```
Perfect numbers between 1 and 100:
6
28
```

3. **RDD for Square Roots between 1 and 20 Code:**

```
from pyspark import SparkContext, SparkConf
import math

# Create a SparkContext
conf = SparkConf().setAppName("SquareRootNumbers").setMaster("local")
sc = SparkContext(conf=conf)

# Create an RDD containing numbers from 1 to 20
numbers_rdd = sc.parallelize(range(1, 21))

# Apply the square root function to each element in the RDD
square_root_rdd = numbers_rdd.map(lambda x: math.sqrt(x))

# Collect and print square roots
square_roots = square_root_rdd.collect()

for square_root in square_roots:
    print(square_root)

# Stop the SparkContext
sc.stop()
```

Output:

```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
3.3166247903554
3.4641016151377544
3.605551275463989
3.7416573867739413
3.872983346207417
4.0
4.123105625617661
4.242640687119285
4.358898943540674
4.47213595499958
```

4 . write a spark program on a list and display the result.

Code:

```
#importing the necessary spark libraries
from pyspark import SparkContext

# Create a SparkContext
sc = SparkContext("local","RDD example")

#creating an rdd from a list of numbers
numbers=[1,2,3,4,5]
rdd = sc.parallelize(numbers)

#performing transformation on rdd
squared_rdd = rdd.map(lambda x: x*x)
filtered_rdd = squared_rdd.filter(lambda x: x > 10)

#performing an action to collect the final result
result = filtered_rdd.collect()

#printing the result
print(result)
```

Output:

```
[16, 25]
```