

Προχωρημένα Θέματα Βάσεων Δεδομένων

Μαρία Κοιλαλού | Μυρτώ Ορφανάκου
AM:03119211 | AM:03119173

16 Ιανουαρίου 2024

1 Εγκατάσταση Apache Spark

Για την εγκατάσταση του Apache Spark ώστε να εκτελείται πάνω από το διαχειριστή πόρων του Apache Hadoop και YARN, δημιουργήσαμε δύο Virtual Machines στην πλατφόρμα okeanos-knossos ακολουθώντας τις οδηγίες που μας δόθηκαν. Οι web εφαρμογές είναι προσβάσιμες και διαμορφωμένες σύμφωνα με τις οδηγίες στα παρακάτω links:

YARN
HDFS
Spark History

Spark Session

Αρχικά, υλοποιήθηκε μια συνάρτηση που δημιουργεί ένα Spark Session και δέχεται ως όρισμα τον αριθμό των executors.

```
1 from pyspark.sql import SparkSession
2
3 def create_spark_session(num_executors):
4
5     spark = SparkSession.builder \
6         .appName("ADV DATABASES") \
7         .config("spark.executor.instances", str(num_executors)) \
8         .config("spark.executor.cores", "1") \
9         .config("spark.executor.memory", "1g") \
10        .config("spark.driver.memory", "4g") \
11        .config("spark.cleaner.periodicGC.interval", "1min") \
12        .config("spark.memory.offHeap.enabled", "true") \
13        .config("spark.memory.offHeap.size", "1g") \
14        .config("spark.default.parallelism", "4") \
15        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
16        .config("spark.dynamicAllocation.enabled", "true") \
17        .config("spark.dynamicAllocation.minExecutors", "2") \
18        .config("spark.dynamicAllocation.maxExecutors", "4") \
19        .getOrCreate()
20
21     return spark
```

Κάθε φορά που θέλουμε να ξεκινήσουμε νέο session καλούμε την συνάρτηση `create_spark_session()` με τον αντίστοιχο αριθμό executors που θέλουμε.

```
1 spark = create_spark_session(4)
2 print("spark session created")
```

2 Create DataFrame

Δημιουργήσαμε ένα schema για το DataFrame του βασικού συνόλου δεδομένων. Τα αρχικά ονόματα των στηλών διατηρήθηκαν, ενώ οι τύποι δεδομένων προσαρμόστηκαν με βάση τα ζητούμενα.

Στη συνέχεια δημιουργήθηκε ένα ενιαίο DataFrame για όλα τα δεδομένα, όπως φαίνεται στον παρακάτω κώδικα:

```
1 schema1 = "`DR_NO` STRING, \  
2 `Date Rptd` STRING, \  
3 `DATE OCC` STRING, \  
4 `TIME OCC` INTEGER, \  
5 `AREA` INTEGER, \  
6 `AREA NAME` STRING, \  
7 `Rpt Dist No` INTEGER, \  
8 `Part 1-2` INTEGER, \  
9 `Crm Cd` INTEGER, \  
10 `Crm Cd Desc` STRING, \  
11 `Mocodes` STRING, \  
12 `Vict Age` INTEGER, \  
13 `Vict Sex` STRING, \  
14 `Vict Descent` STRING, \  
15 `Premis Cd` INTEGER, \  
16 `Premis Desc` STRING, \  
17 `Weapon Used Cd` INTEGER, \  
18 `Weapon Desc` STRING, \  
19 `Status` STRING, \  
20 `Status Desc` STRING, \  
21 `Crm Cd 1` INTEGER, \  
22 `Crm Cd 2` INTEGER, \  
23 `Crm Cd 3` INTEGER, \  
24 `Crm Cd 4` INTEGER, \  
25 `LOCATION` STRING, \  
26 `Cross Street` STRING, \  
27 `LAT` DOUBLE, \  
28 `LON` DOUBLE"  
29  
30  
31 data1 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/crime_data_2010.csv", header=True, schema=schema1)  
32 data2 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/crime_data_2020.csv", header=True, schema=schema1)  
33  
34 df = data1.union(data2).distinct()  
35  
36 df = df.withColumn("Date Rptd", to_date(col("Date Rptd"), "MM/dd/yyyy hh:mm:ss a")) \  
37 .withColumn("DATE OCC", to_date(col("DATE OCC"), "MM/dd/yyyy hh:mm:ss a"))  
38  
39 df.count()  
40 print(f"Total number of rows: {df.count()}")  
41  
42 df.printSchema()
```

Ο συνολικός αριθμός γραμμών του συνόλου δεδομένων, καθώς και ο τύπος κάθε στήλης είναι τα εξής:

```
spark session created  
Total number of rows: 2913595  
root  
|-- DR_NO: string (nullable = true)  
|-- Date Rptd: date (nullable = true)  
|-- DATE OCC: date (nullable = true)  
|-- TIME OCC: integer (nullable = true)  
|-- AREA: integer (nullable = true)  
|-- AREA NAME: string (nullable = true)  
|-- Rpt Dist No: integer (nullable = true)  
|-- Part 1-2: integer (nullable = true)  
|-- Crm Cd: integer (nullable = true)  
|-- Crm Cd Desc: string (nullable = true)  
|-- Mocodes: string (nullable = true)  
|-- Vict Age: integer (nullable = true)  
|-- Vict Sex: string (nullable = true)
```

```

|-- Vict Descent: string (nullable = true)
|-- Premis Cd: integer (nullable = true)
|-- Premis Desc: string (nullable = true)
|-- Weapon Used Cd: integer (nullable = true)
|-- Weapon Desc: string (nullable = true)
|-- Status: string (nullable = true)
|-- Status Desc: string (nullable = true)
|-- Crm Cd 1: integer (nullable = true)
|-- Crm Cd 2: integer (nullable = true)
|-- Crm Cd 3: integer (nullable = true)
|-- Crm Cd 4: integer (nullable = true)
|-- LOCATION: string (nullable = true)
|-- Cross Street: string (nullable = true)
|-- LAT: double (nullable = true)
|-- LON: double (nullable = true)

```

3 Query 1

Link για το Spark UI: Query1 & Query2

Υλοποιήθηκε μία συνάρτηση για το Query 1 χρησιμοποιώντας DataFrame API:

Dataframe API

```

1 def query1_df(df):
2     crime_date = df.withColumn("Year", year("DATE OCC")).withColumn("Month", month("DATE OCC"))
3
4     count = crime_date.groupBy("Year", "Month").count()
5
6     window_spec = Window.partitionBy("Year").orderBy(desc("count"))
7     top_months = count.withColumn("rank", dense_rank().over(window_spec)).filter(col("rank") <= 3)
8
9     top_months = top_months.orderBy("Year", "rank")
10
11    return top_months

```

Στη συνέχεια το Query 1 υλοποιήθηκε με SQL API:

SQL API

```

1 def query1_sql(df):
2     crime_date = df.withColumn("Year", year("DATE OCC")).withColumn("Month", month("DATE OCC"))
3
4     # Δημιουργία προσωρινής προβολής
5     crime_date.createOrReplaceTempView("crimes")
6
7     # SQL ερώτημα για την εύρεση των τριών μηνών με τον υψηλότερο αριθμό εγκλημάτων ανά έτος
8     query1 = """
9     SELECT Year, Month, count, rank
10    FROM (
11        SELECT Year, Month, count(*) AS count,
12               DENSE_RANK() OVER (PARTITION BY Year ORDER BY count(*) DESC) AS rank
13        FROM crimes
14        GROUP BY Year, Month
15    )
16    WHERE rank <= 3
17    ORDER BY Year, rank
18    """
19
20    top_months = crime_date.sparkSession.sql(query1)
21
22    return top_months

```

Παρακάτω φαίνονται τα αποτελέσματα του Query 1 για τις δύο διαφορετικές υλοποιήσεις, καθώς και οι αντίστοιχοι χρόνοι εκτέλεσης. Παρατηρούμε ότι υπάρχει διαφορά στην επίδοση των δύο APIs. Συγκεκριμένα, το DataFrame API πραγματοποιήθηκε σε πολύ λιγότερο χρόνο σε σχέση με το SQL API.

| year | month | crime_total | # |
|--|-------|-------------|---|
| 2010 | 1 | 19515 | 1 |
| 2010 | 3 | 18131 | 2 |
| 2010 | 7 | 17856 | 3 |
| 2011 | 1 | 18134 | 1 |
| 2011 | 7 | 17283 | 2 |
| 2011 | 10 | 17034 | 3 |
| 2012 | 1 | 17943 | 1 |
| 2012 | 8 | 17661 | 2 |
| 2012 | 5 | 17502 | 3 |
| 2013 | 8 | 17440 | 1 |
| 2013 | 1 | 16820 | 2 |
| 2013 | 7 | 16644 | 3 |
| 2014 | 7 | 12196 | 1 |
| 2014 | 10 | 12133 | 2 |
| 2014 | 8 | 12028 | 3 |
| 2015 | 10 | 19219 | 1 |
| 2015 | 8 | 19011 | 2 |
| 2015 | 7 | 18709 | 3 |
| 2016 | 10 | 19659 | 1 |
| 2016 | 8 | 19490 | 2 |
| only showing top 20 rows | | | |
| Q1 Dataframe time: 0.537975549697876 seconds | | | |

| year | month | crime_total | # |
|--|-------|-------------|---|
| 2010 | 1 | 19515 | 1 |
| 2010 | 3 | 18131 | 2 |
| 2010 | 7 | 17856 | 3 |
| 2011 | 1 | 18134 | 1 |
| 2011 | 7 | 17283 | 2 |
| 2011 | 10 | 17034 | 3 |
| 2012 | 1 | 17943 | 1 |
| 2012 | 8 | 17661 | 2 |
| 2012 | 5 | 17502 | 3 |
| 2013 | 8 | 17440 | 1 |
| 2013 | 1 | 16820 | 2 |
| 2013 | 7 | 16644 | 3 |
| 2014 | 7 | 12196 | 1 |
| 2014 | 10 | 12133 | 2 |
| 2014 | 8 | 12028 | 3 |
| 2015 | 10 | 19219 | 1 |
| 2015 | 8 | 19011 | 2 |
| 2015 | 7 | 18709 | 3 |
| 2016 | 10 | 19659 | 1 |
| 2016 | 8 | 19490 | 2 |
| only showing top 20 rows | | | |
| Q1 SQL time: 0.861966609954834 seconds | | | |

4 Query 2

DataFrame\SQL API

Υλοποιήθηκε μία συνάρτηση για το Query 2 χρησιμοποιώντας DataFrame/ SQL API:

```

1 def query2_df(df):
2
3
4 def day_part(hour):
5     if 500 <= hour < 1200:
6         return "Πρωί"
7     elif 1200 <= hour < 1700:
8         return "Απόγευμα"
9     elif 1700 <= hour < 2100:
10        return "Βράδυ"
11    else:
12        return "Νύχτα"
13
14 day_part_udf = udf(day_part, StringType())
15
16 df_day_part = df.withColumn("DayPart", day_part_udf(col("TIME OCC")))
17
18 df_street_crimes = df_day_part.filter(col("Premis Desc") == "STREET").groupBy("DayPart").count().orderBy(col("count").desc())
19
20 return df_street_crimes

```

RDD API

Στη συνέχεια το Query 2 υλοποιήθηκε με RDD API:

```
1 def query2_rdd(df):
2
3 def day_part(hour):
4     if 500 <= hour < 1200:
5         return "Πρωί"
6     elif 1200 <= hour < 1700:
7         return "Απόγευμα"
8     elif 1700 <= hour < 2100:
9         return "Βράδυ"
10    else:
11        return "Νύχτα"
12
13 rdd = df.rdd.filter(lambda row: row['Premis Desc'] == 'STREET')
14
15 def map_day_part(record):
16     hour = int(record["TIME OCC"])
17     part = day_part(hour)
18     return (part, 1)
19
20 rdd_mapped = rdd.map(map_day_part)
21 rdd_reduced = rdd_mapped.reduceByKey(lambda a, b: a + b)
22
23 rdd_street_crimes = rdd_reduced.sortBy(lambda x: x[1], ascending=False)
24
25 return rdd_street_crimes
```

Ακολουθούν τα αποτελέσματα του Query 2 για τις δύο διαφορετικές υλοποιήσεις, καθώς και οι αντίστοιχοι χρόνοι εκτέλεσης. Παρατηρούμε ότι υπάρχει διαφορά στην επίδοση των δύο APIs. Συγκεκριμένα, το DataFrame/ SQL API πραγματοποιήθηκε σε πολύ λιγότερο χρόνο σε σχέση με το RDD API.

| DayPart | count |
|---|--------|
| Νύχτα | 231546 |
| Βράδυ | 182141 |
| Απόγευμα | 143974 |
| Πρωί | 120358 |
| only showing top 4 rows | |
| Q2 Dataframe time: 0.2953650951385498 seconds | |

RDD: [(‘Νύχτα’, 231546), (‘Βράδυ’, 182141), (‘Απόγευμα’, 143974), (‘Πρωί’, 120358)]
Q2 RDD time: 61.14855885505676 seconds.

5 Query 3

Link για το Spark UI: Query3: Exec 2 Query3: Exec 3 Query4: Exec 4

Στο συγκεκριμένο ερώτημα χρησιμοποιήθηκαν τα δευτερεύοντα σύνολα δεδομένων [revgecoding](#) και [LA_income_2015](#). Επιπλέον, πραγματοποιήθηκε map για τα Vict Descent.

```
1 data3 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/LA_income_2015.csv", header=True, schema=schema2)
2 data4 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/revgecoding.csv", header=True, schema=schema3)
3
4 schema3 = "`LAT` DOUBLE, \
5           `LON` DOUBLE, \
6           `ZIPcode` INTEGER"
7
8 schema2 = "`Zip Code` INTEGER, \
9           `Community` STRING, \
```

```

10     `Estimated Median Income` STRING"
11
12 descent_mapping = {
13     'A': 'Other Asian',
14     'B': 'Black',
15     'C': 'Chinese',
16     'D': 'Cambodian',
17     'F': 'Filipino',
18     'G': 'Guamanian',
19     'H': 'Hispanic/Latin/Mexican',
20     'I': 'American Indian/Alaskan Native',
21     'J': 'Japanese',
22     'K': 'Korean',
23     'L': 'Laotian',
24     'O': 'Other',
25     'P': 'Pacific Islander',
26     'S': 'Samoan',
27     'U': 'Hawaiian',
28     'V': 'Vietnamese',
29     'W': 'White',
30     'X': 'Unknown',
31     'Z': 'Asian Indian'
32 }
33
34 data3 = data3.withColumn("Estimated Median Income", regexp_replace(col("Estimated Median Income"), "\\$", ""))
35 data3 = data3.withColumn("Estimated Median Income", regexp_replace(col("Estimated Median Income"), ",", "").cast("float"))
36
37 crime_year = df.withColumn("Year", year("DATE OCC"))
38
39 crime_2015 = crime_year.filter(
40     (col("Year") == 2015) &
41     (col("Vict Descent").isNotNull()))
42
43 def map_descent(code):
44     return descent_mapping.get(code, "Unknown") # Default to "Unknown" if code not found
45
46 map_descent_udf = udf(map_descent, StringType())
47
48 crime_2015 = crime_2015.withColumn("Vict Descent", map_descent_udf(crime_2015["Vict Descent"]))
49
50 revgecoding = data4.dropDuplicates(['LAT', 'LON'])

```

Έπειτα υλοποιήθηκε το Query 3 χρησιμοποιώντας DataFrame/ SQL API.

```

1 def query3 (crime_2015, data3, revgecoding):
2
3     crime_zip = crime_2015.join(revgecoding, ["LAT", "LON"], "left")
4
5     best3_zip = data3.orderBy("Estimated Median Income", ascending=False).limit(3)
6     worst3_zip = data3.orderBy("Estimated Median Income", ascending=True).limit(3)
7
8     best3_zip_list = [row['Zip Code'] for row in best3_zip.collect()]
9     worst3_zip_list = [row['Zip Code'] for row in worst3_zip.collect()]
10
11     crimes = crime_zip.filter(
12         (col("ZIPcode").isin(best3_zip_list)) |
13         (col("ZIPcode").isin(worst3_zip_list))
14     )
15
16     vict_descent_count = crimes.groupBy("Vict Descent").count().orderBy("count", ascending=False)
17
18
19     return vict_descent_count

```

Το Query 3 εκτελέστηκε σε ένα for loop, δημιουργώντας διαδοχικά τρία Spark Sessions για 2, 3 και 4 executors. Παρακάτω φαίνονται τα αποτελέσματα και η διάρκεια της κάθε εκτέλεσης αντίστοιχα. Παρατηρούμε ότι ταχύτερα εκτελέστηκε το Query με 3 executors, έπειτα με 2 και τέλος με 4.

| Vict Descent | count |
|-------------------------------------|-------|
| Hispanic/Latin/Mexican | 1053 |
| White | 610 |
| Black | 349 |
| Other | 272 |
| Unknown | 71 |
| Other Asian | 46 |
| Korean | 4 |
| Chinese | 1 |
| American Indian/Alaskan Native | 1 |
| Number of Executors: 2 | |
| Q3 time: 12.225924015045166 seconds | |

| Vict Descent | count |
|-------------------------------------|-------|
| Hispanic/Latin/Mexican | 1053 |
| White | 610 |
| Black | 349 |
| Other | 272 |
| Unknown | 71 |
| Other Asian | 46 |
| Korean | 4 |
| Chinese | 1 |
| American Indian/Alaskan Native | 1 |
| Number of Executors: 3 | |
| Q3 time: 11.629308462142944 seconds | |

| Vict Descent | count |
|-------------------------------------|-------|
| Hispanic/Latin/Mexican | 1053 |
| White | 610 |
| Black | 349 |
| Other | 272 |
| Unknown | 71 |
| Other Asian | 46 |
| Korean | 4 |
| Chinese | 1 |
| American Indian/Alaskan Native | 1 |
| Number of Executors: 4 | |
| Q3 time: 18.245003938674927 seconds | |

6 Query 4

Link για το Spark UI: Query 4

Για το ζητούμενο 6 χρησιμοποιήθηκαν επιπλέον τα δεδομένα [LAPD_Police_Stations](#). Το Query 4 υλοποιήθηκε με DataFrame/SQL API και εκτελέστηκε με 4 executors. Αρχικά δημιουργήθηκε μια συνάρτηση για τον υπολογισμό της απόστασης μεταξύ 2 σημείων με συγκεκριμένες συντεταγμένες. Στη συνέχεια πραγματοποιήθηκε join μεταξύ του αρχικού Dataframe και του [LAPD_Police_Stations](#). Έπειτα, γία την περίπτωση του κοντινότερου station, πραγματοποιήθηκε ένα cross join μεταξύ των δύο dataframe για τον εντοπισμό του κοντινότερου τμήματος σε κάθε έγκλημα.

```

1 schema4 = "`X` DOUBLE, \
2             `Y` DOUBLE, \
3             `FID` INTEGER, \
4             `DIVISION` STRING, \
5             `LOCATION` STRING, \
6             `PREC` INTEGER"
7
8 data5 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/LAPD_Police_Stations.csv", header=True, schema=schema4)

```

```

1 def query4(df, data5):
2
3     def haversine(lat1, lon1, lat2, lon2):
4         # Radius of the Earth in kilometers
5         R = 6371.0
6
7         lat1_rad = math.radians(lat1)
8         lon1_rad = math.radians(lon1)
9         lat2_rad = math.radians(lat2)
10        lon2_rad = math.radians(lon2)
11
12        dlat = lat2_rad - lat1_rad
13        dlon = lon2_rad - lon1_rad
14
15        a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
16        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
17
18        distance = R * c
19        return distance
20
21    def get_distance(lat1, long1, lat2, long2):
22
23        def is_valid_coordinate(lat, lon):
24            return -90 <= lat <= 90 and -180 <= lon <= 180
25
26
27        if not is_valid_coordinate(lat1, long1) or not is_valid_coordinate(lat2, long2):
28            # Print the invalid rows
29            print(f"Invalid row: lat1={lat1}, long1={long1}, lat2={lat2}, long2={long2}")
30            return -1
31
32        try:
33            return haversine(lat1, long1, lat2, long2)
34        except ValueError:
35            return -1
36
37    df = df.filter(
38        (df["AREA NAME"] != "Null Island") &
39        (df["Weapon Used Cd"].substr(1, 1) == "1")
40    )
41
42    joined_df = df.join(data5, df["AREA"] == data5["PREC"])
43
44    distance_udf = udf(get_distance)
45
46    distance_df = joined_df.withColumn(
47        "DISTANCE",
48        distance_udf(
49            F.col("LAT"), F.col("LON"),
50            F.col("Y"), F.col("X")
51        ).cast("double")
52    )
53
54    query_4_1a = distance_df.groupBy("Year").agg(
55        F.avg("DISTANCE").alias("average_distance"),
56        F.count("#").alias("#")
57    ).orderBy("Year").withColumnRenamed("Year", "year")
58
59    query_4_1b = distance_df.groupBy("DIVISION").agg(
60        F.avg("DISTANCE").alias("average_distance"),
61        F.count("#").alias("#")
62    ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")
63
64    print Απόσταση(" από το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό:")
65    print("(a)")
66    query_4_1a.show()
67    print("(b)")
68    query_4_1b.show()
69
70    cross_joined_df = df.crossJoin(data5.withColumnRenamed("LAT", "Y").withColumnRenamed("LON", "X"))

```



```

71 cross_joined_df = cross_joined_df.withColumn(
72     "DISTANCE",
73     distance_udf(col("LAT"), col("LON"), col("Y"), col("X")).cast("double")
74 )
75
76 windowSpec = Window.partitionBy("DR_NO").orderBy("DISTANCE")
77
78 nearest_station_df = cross_joined_df.withColumn(
79     "row_num",
80     F.row_number().over(windowSpec)
81 ).filter(col("row_num") == 1).drop("row_num")
82
83 query_4_2a = nearest_station_df.groupBy("Year").agg(
84     F.avg("DISTANCE").alias("average_distance"),
85     F.count("#").alias("#")
86 ).orderBy("Year").withColumnRenamed("Year", "year")
87
88 query_4_2b = nearest_station_df.groupBy("DIVISION").agg(
89     F.avg("DISTANCE").alias("average_distance"),
90     F.count("#").alias("#")
91 ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")
92
93
94
95 printΑπόσταση(" από το πλησιέστερο αστυνομικό τμήμα:")
96 print("(a)")
97 query_4_2a.show()
98 print("(b)")
99 query_4_2b.show()

```

Τα αποτελέσματα της εκτέλεσης για τα 4 ερωτήματα είναι τα εξής:

Απόσταση από το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό:

| year | average_distance | # |
|------|--------------------|-------|
| 2010 | 4.315547525861609 | 8213 |
| 2011 | 2.7931783031826134 | 7232 |
| 2012 | 37.401521647671 | 6550 |
| 2013 | 2.826412721201962 | 5838 |
| 2014 | 11.631025289489838 | 4230 |
| 2015 | 2.706097992762391 | 6763 |
| 2016 | 2.7176445421299724 | 8100 |
| 2017 | 5.955847913803835 | 7788 |
| 2018 | 2.732823649229879 | 7413 |
| 2019 | 2.739941972172148 | 7129 |
| 2020 | 8.614767812336167 | 8491 |
| 2021 | 30.97834129556093 | 9767 |
| 2022 | 2.6086561864507893 | 10025 |
| 2023 | 2.555141057454313 | 8741 |

| division | average_distance | # |
|-----------------|--------------------|-------|
| 77TH STREET | 5.736614947109006 | 16546 |
| SOUTHEAST | 9.578741738383359 | 11782 |
| NEWTON | 9.865416685211947 | 9613 |
| SOUTHWEST | 4.15636383556516 | 8625 |
| HOLLENBECK | 14.994438060689776 | 6111 |
| HARBOR | 13.360482218365267 | 5431 |
| RAMPART | 4.098521839067684 | 4989 |
| MISSION | 7.743899200430639 | 4153 |
| OLYMPIC | 1.827684160849825 | 3971 |
| NORTHEAST | 10.43910354785769 | 3846 |
| HOLLYWOOD | 12.080122049355651 | 3551 |
| FOOTHILL | 3.8148915583594225 | 3484 |
| CENTRAL | 4.763802684561787 | 3466 |
| WILSHIRE | 13.350395954999458 | 3422 |
| NORTH HOLLYWOOD | 14.087690925056263 | 3321 |
| WEST VALLEY | 17.084643689509413 | 2786 |
| PACIFIC | 13.244049319509951 | 2647 |
| VAN NUYS | 2.2172720177483716 | 2645 |
| DEVONSHIRE | 15.049134124450779 | 2280 |
| TOPANGA | 3.488714475764334 | 2101 |

Απόσταση από το πλησιέστερο αστυνομικό τμήμα:

| year | average_distance | # |
|------|--------------------|-------|
| 2010 | 3.9654805060979808 | 8213 |
| 2011 | 2.4618188856645915 | 7232 |
| 2012 | 37.04806556244542 | 6550 |
| 2013 | 2.456180337945913 | 5838 |
| 2014 | 11.240705060052049 | 4230 |
| 2015 | 2.387902781763031 | 6763 |
| 2016 | 2.4291509215379383 | 8100 |
| 2017 | 5.620278866952368 | 7788 |
| 2018 | 2.4090835060969624 | 7413 |
| 2019 | 2.4301661049761196 | 7129 |
| 2020 | 8.305664894299348 | 8491 |
| 2021 | 30.666116941658995 | 9767 |
| 2022 | 2.312967928245974 | 10025 |
| 2023 | 2.2716948056968684 | 8741 |

| division | average_distance | # |
|-----------------|--------------------|-------|
| 77TH STREET | 1.7215717802940704 | 13314 |
| SOUTHWEST | 2.281362128260118 | 11195 |
| SOUTHEAST | 2.210009298415925 | 10836 |
| NEWTON | 1.5697887030696482 | 7150 |
| WILSHIRE | 2.443640741448432 | 6227 |
| HOLLENBECK | 103.76094896583089 | 6215 |
| HOLLYWOOD | 2.0025711504573533 | 5328 |
| HARBOR | 3.905879971758679 | 5305 |
| OLYMPIC | 1.6650515588081933 | 5071 |
| RAMPART | 1.3976311372228347 | 4677 |
| VAN NUYS | 2.9536720802889667 | 4587 |
| FOOTHILL | 3.612771771528315 | 4214 |
| CENTRAL | 1.0231066779455342 | 3597 |
| NORTH HOLLYWOOD | 2.721425990200832 | 3270 |
| NORTHEAST | 3.7517940872523634 | 3093 |
| WEST VALLEY | 2.7951039642375455 | 2716 |
| MISSION | 3.8087595625812636 | 2625 |
| PACIFIC | 3.700480667929206 | 2521 |
| TOPANGA | 3.0254147394551283 | 2146 |
| DEVONSHIRE | 2.9876944488748034 | 1180 |

7 hint & explain

Εκτελέσαμε το Query 3 σε ένα for loop που επιλέγει μια διαφορετική στρατηγική join σε κάθε επανάληψη.

Query 3

Τα αποτελέσματα της εκτέλεσης φαίνονται στον ακόλουθο σύνδεσμο. Ταχύτερα αποτελέσματα παρουσιάζει η στρατηγική Shuffle Hash. Αυτό συμβαίνει επειδή τα κλειδιά συγχώνευσης έχουν ανομοιογενή κατανομή, οπότε η "Shuffle Hash" μπορεί να κατανείμει τα δεδομένα πιο ομοιόμορφα σε κομμάτια, ιδίως όταν αντιμετωπίζουμε μεγάλα σύνολα δεδομένων, μειώνοντας την επίδραση της ανομοιογένειας δεδομένων και βελτιώνοντας την απόδοση.

| JOIN STRATEGY | DURATION |
|----------------------|----------|
| BROADCAST | 35s |
| MERGE | 21s |
| SHUFFLE HASH | 19s |
| SHUFFLE REPLICATE NL | 28s |

Link για το Spark UI: Query3 Hint & Explain

```
1 join_strategies = ["broadcast", "merge", "shuffle_hash", "shuffle_replicate_nl"]
2
3 for strategy in join_strategies:
4     print(f"Executing query with {strategy} join strategy")
5     query3_hne_results = query3_hne.query3(crime_2015, data3, revgecoding, strategy)
6     query3_hne_results.show()
```

Στον κώδικα για το Query 3 έχουμε προσθέσει τα hint και explain.

```
1 def query3(crime_2015, data3, revgecoding, join_strategy):
2
3     crime_zip = crime_2015.join(revgecoding.hint(join_strategy), ["LAT", "LON"], "left")
4
5     best3_zip = data3.orderBy("Estimated Median Income", ascending=False).limit(3)
6     worst3_zip = data3.orderBy("Estimated Median Income", ascending=True).limit(3)
7
8     best3_zip_list = [row['Zip Code'] for row in best3_zip.collect()]
9     worst3_zip_list = [row['Zip Code'] for row in worst3_zip.collect()]
10
11     crimes = crime_zip.filter(
12         (col("ZIPcode").isin(best3_zip_list)) |
13         (col("ZIPcode").isin(worst3_zip_list))
14     )
15
16     vict_descent_count = crimes.groupBy("Vict Descent").count().orderBy("count", ascending=False)
17
18     vict_descent_count.explain()
19
20     return vict_descent_count
```

Query 4

Ο κώδικας για το Query 4 έχει συμπληρωθεί με τα απαραίτητα hint και explain. Τα αποτελέσματα της εκτέλεσης φαίνονται στους ακόλουθους συνδέσμους.

Για το 4_1a ερώτημα καλύτερη στρατηγική για το join είναι η Merge, ενώ για το 4_1b ερώτημα είναι η Broadcast. Οι χρόνοι του ερωτήματος (a) είναι σημαντικά μεγαλύτεροι από αυτούς του (b) διότι λόγω lazy evaluation από το spark το ερώτημα (a) εκτελεί κομμάτια κώδικα που το (b) βρίσκει έτοιμο κατά την εκτέλεση του.

| | JOIN STRATEGY | DURATION |
|-----|----------------------|----------|
| (a) | BROADCAST | 36s |
| | MERGE | 35s |
| | SHUFFLE HUSH | 36s |
| | SHUFFLE REPLICATE NL | 52s |

| | JOIN STRATEGY | DURATION |
|-----|----------------------|----------|
| (b) | BROADCAST | 14s |
| | MERGE | 15s |
| | SHUFFLE HUSH | 18s |
| | SHUFFLE REPLICATE NL | 45s |

Όσον αφορά στο 4_1a, αυτό συμβαίνει επειδή το join γίνεται με βάση το Year και τα σύνολα δεδομένων είναι σχετικά μεγάλα. Η Merge περιλαμβάνει την ταξινόμηση και στη συνέχεια τη συγχώνευση των συνόλων δεδομένων, η οποία είναι αποδοτική για μεγάλα σύνολα δεδομένων. Επιπλέον, τα δεδομένα είναι ομοιόμορφα κατανομημένα σε κομμάτια για το κλειδί συγχώνευσης Year, καθιστώντας τη στρατηγική Merge βέλτιστη. Σχετικά με το 4_1b, ο πίνακας διαστάσεων (data5) είναι σχετικά μικρός, οπότε το Spark μπορεί να τον μεταδώσει αποδοτικά σε όλους τους κόμβους, μειώνοντας την ανάγκη για μεταφορά δεδομένων σε όλο το δίκτυο. Η μετάδοση είναι ιδιαίτερα αποδοτική όταν ένας από τους πίνακες στη συγχώνευση είναι αρκετά μικρός ώστε να χωρέσει στη μνήμη κάθε κόμβου. Επιπλέον, η στρατηγική join Broadcast είναι αποδοτική όταν πραγματοποιείται join με βάση την ισότητα, όπως στην περίπτωση του πεδίου DIVISION.

Query 4 Broadcast Query 4 Merge Query 4 Shuffle Hush Query 4 Shuffle Replicate

```
1 def query4(df, data5, join_strategy):
2
3     def haversine(lat1, lon1, lat2, lon2):
4         # Radius of the Earth in kilometers
5         R = 6371.0
6
7         lat1_rad = math.radians(lat1)
8         lon1_rad = math.radians(lon1)
9         lat2_rad = math.radians(lat2)
10        lon2_rad = math.radians(lon2)
11
12        dlat = lat2_rad - lat1_rad
13        dlon = lon2_rad - lon1_rad
```

```

14 a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
15 c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
16
17 distance = R * c
18 return distance
19
20
21 def get_distance(lat1, long1, lat2, long2):
22
23     def is_valid_coordinate(lat, lon):
24         return -90 <= lat <= 90 and -180 <= lon <= 180
25
26
27     if not is_valid_coordinate(lat1, long1) or not is_valid_coordinate(lat2, long2):
28         # Print the invalid rows
29         print(f"Invalid row: lat1={lat1}, long1={long1}, lat2={lat2}, long2={long2}")
30         return -1
31
32     try:
33         return haversine(lat1, long1, lat2, long2)
34     except ValueError:
35         return -1
36
37 df = df.filter(
38     (df["AREA NAME"] != "Null Island") &
39     (df["Weapon Used Cd"].substr(1, 1) == "1")
40 )
41
42 joined_df = df.join(data5.hint(join_strategy), df["AREA"] == data5["PREC"])
43
44 joined_df.explain()
45
46 distance_udf = udf(get_distance)
47
48 distance_df = joined_df.withColumn(
49     "DISTANCE",
50     distance_udf(
51         F.col("LAT"), F.col("LON"),
52         F.col("Y"), F.col("X")
53     ).cast("double")
54 )
55
56 query_4_1a = distance_df.groupBy("Year").agg(
57     F.avg("DISTANCE").alias("average_distance"),
58     F.count("#").alias("#")
59 ).orderBy("Year").withColumnRenamed("Year", "year")
60
61 query_4_1b = distance_df.groupBy("DIVISION").agg(
62     F.avg("DISTANCE").alias("average_distance"),
63     F.count("#").alias("#")
64 ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")
65
66 print("Απόσταση από το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό:")
67 print("(a)")
68 query_4_1a.show()
69 print("(b)")
70 query_4_1b.show()
71
72 cross_joined_df = df.crossJoin(data5.withColumnRenamed("LAT", "Y").withColumnRenamed("LON", "X"))
73
74 cross_joined_df = cross_joined_df.withColumn(
75     "DISTANCE",
76     distance_udf(col("LAT"), col("LON"), col("Y"), col("X")).cast("double")
77 )
78
79 windowSpec = Window.partitionBy("DR_NO").orderBy("DISTANCE")
80
81 nearest_station_df = cross_joined_df.withColumn(
82     "row_num",
83     F.row_number().over(windowSpec)
84 ).filter(col("row_num") == 1).drop("row_num")

```

```

85 query_4_2a = nearest_station_df.groupby("Year").agg(
86     F.avg("DISTANCE").alias("average_distance"),
87     F.count("*").alias("#")
88 ).orderBy("Year").withColumnRenamed("Year", "year")
89
90 query_4_2b = nearest_station_df.groupby("DIVISION").agg(
91     F.avg("DISTANCE").alias("average_distance"),
92     F.count("*").alias("#")
93 ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")
94
95
96
97 print("Απόσταση από το πλησιέστερο αστυνομικό τμήμα:")
98 print("(a)")
99 query_4_2a.show()
100 print("(b)")
101 query_4_2b.show()

```