Προχωρημένα Θέματα Βάσεων Δεδομένων

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

18 Ιανουαρίου 2024

Όλα τα αρχεία υπάρχουν στο GitHub στο link: GitHub Repository

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 παρακάτων links:

YARN
HDFS
Spark History

Spark Session

Ο κώδικας βρίσκεται στο /scripts/session.py
Αρχικά, υλοποιήθηκε μια συνάρτηση που δημιουργεί ένα Spark Session και δέχεται ως όρισμα τον αριθμό των executors.

```
from pyspark.sql import SparkSession

def create_spark_session(num_executors):

    spark = SparkSession.builder \
        .appName("ADV DATABASES") \
        .config("spark.executor.instances", str(num_executors)) \
        .config("spark.executor.cores", "1") \
        .config("spark.executor.memory", "1g") \
        .config("spark.driver.memory", "4g") \
        .config("spark.cleaner.periodicGC.interval", "1min") \
        .config("spark.memory.offHeap.enabled", "true") \
        .config("spark.memory.offHeap.size", "1g") \
        .config("spark.default.parallelism", "4") \
        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
        .config("spark.dynamicAllocation.enabled", "true") \
        .config("spark.dynamicAllocation.minExecutors", "2") \
        .config("spark.dynamicAllocation.maxExecutors", "4") \
        .getOrCreate()

    return spark
```

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

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

Για να τρέξουμε τον κώδικα εκτελούμε sparksubmit /scripts/main.py .Εκτελούνται έτσι οι κώδικες για όλα τα queries.
Ο κώδικας βρίσκεται στο /scripts/main.py

2 Create DataFrame

Το output βρίσκεται στο /output/output1.py

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:

```
    schema1 = "`DR_NO` STRING, \
               `Date Rptd` STRING, \
               `DATE OCC` STRING, \
               `TIME OCC` INTEGER, \
               `AREA` INTEGER, \
               `AREA NAME` STRING, \
               `Rpt Dist No` INTEGER, \
               `Part 1-2` INTEGER, \
               `Crm Cd` INTEGER, \
               `Crm Cd Desc` STRING, \
               `Mocodes` STRING, \
               `Vict Age` INTEGER, \
               `Vict Sex` STRING, \
               `Vict Descent` STRING, \
               `Premis Cd` INTEGER, \
               `Premis Desc` STRING, \
               `Weapon Used Cd` INTEGER, \
               `Weapon Desc` STRING, \
               `Status` STRING, \
               `Status Desc` STRING, \
               `Crm Cd 1` INTEGER, \
               `Crm Cd 2` INTEGER, \
               `Crm Cd 3` INTEGER, \
               `Crm Cd 4` INTEGER, \
               `LOCATION` STRING, \
               `Cross Street` STRING, \
               `LAT` DOUBLE, \
               `LON` DOUBLE"


    data1 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/crime_data_2010.csv", header=True, schema=schema1)
    data2 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/crime_data_2020.csv", header=True, schema=schema1)

    df = data1.union(data2).distinct()

    df = df.withColumn("Date Rptd", to_date(col("Date Rptd"), "MM/dd/yyyy hh:mm:ss a")) \
    .withColumn("DATE OCC", to_date(col("DATE OCC"), "MM/dd/yyyy hh:mm:ss a"))

    df.count()
    print(f"Total number of rows: {df.count()}")

    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

Ο κώδικας για την εκτέλεση του query βρίσκεται στο /scripts/queries/query1.py το οποίο καλεί η main κατά την εκτέλεση της
Το output του query 1 βρίσκεται στο /output/output2.py
Link για το Spark UI: Query1 & Query2

Υλοποιήθηκε μία συνάρτηση για το Query 1 χρησιμοποιώντας DataFrame API:

Dataframe API

```python
def query1_df(df):
crime_date = df.withColumn("Year", year("DATE OCC")).withColumn("Month", month("DATE OCC"))

count = crime_date.groupBy("Year", "Month").count()

window_spec = Window.partitionBy("Year").orderBy(desc("count"))
top_months = count.withColumn("rank", dense_rank().over(window_spec)).filter(col("rank") <= 3)

top_months = top_months.orderBy("Year", "rank")

return top_months
```

Στη συνέχεια το Query 1 υλοποιήθηκε με SQL API:

SQL API

```python
def query1_sql(df):
crime_date = df.withColumn("Year", year("DATE OCC")).withColumn("Month", month("DATE OCC"))

# Δημιουργία προσωρινής προβολής
crime_date.createOrReplaceTempView("crimes")

# SQL ερώτημα για την εύρεση των τριών μηνών με τον υψηλότερο αριθμό εγκλημάτων ανά έτος
query1 = """
SELECT Year, Month, count, rank
FROM (
```

```
        SELECT Year, Month, count(*) AS count,
               DENSE_RANK() OVER (PARTITION BY Year ORDER BY count(*) DESC) AS rank
        FROM crimes
        GROUP BY Year, Month
    )
    WHERE rank <= 3
    ORDER BY Year, rank
    """

    top_months = crime_date.sparkSession.sql(query1)

    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

Ο κώδικας για την εκτέλεση του query βρίσκεται στο /scripts/queries/query2.py το οποίο καλεί η main κατά την εκτέλεση της
Το output του query 2 βρίσκεται στο /output/output3.py

DataFrame\SQL API

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

```

    def query2_df(df):

    def day_part(hour):
        if 500 <= hour < 1200:
            return "Πρωί"
```

```
        elif 1200 <= hour < 1700:
            return "Απόγευμα"
        elif 1700 <= hour < 2100:
            return "Βράδυ"
        else:
            return "Νύχτα"

    day_part_udf = udf(day_part, StringType())

    df_day_part = df.withColumn("DayPart", day_part_udf(col("TIME OCC")))

    df_street_crimes = df_day_part.filter(col("Premis Desc") == "STREET").groupBy("DayPart").count().orderBy(col("count").
    desc())

    return df_street_crimes
```

RDD API

Στη συνέχεια το Query 2 υλοποιήθηκε με RDD API:

```
    def query2_rdd(df):

    def day_part(hour):
        if 500 <= hour < 1200:
            return "Πρωί"
        elif 1200 <= hour < 1700:
            return "Απόγευμα"
        elif 1700 <= hour < 2100:
            return "Βράδυ"
        else:
            return "Νύχτα"

    rdd = df.rdd.filter(lambda row: row['Premis Desc'] == 'STREET')

    def map_day_part(record):
        hour = int(record["TIME OCC"])
        part = day_part(hour)
        return (part, 1)

    rdd_mapped = rdd.map(map_day_part)
    rdd_reduced = rdd_mapped.reduceByKey(lambda a, b: a + b)

    rdd_street_crimes = rdd_reduced.sortBy(lambda x: x[1], ascending=False)

    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

Ο κώδικας βρίσκεται στο /scripts/queries/query3.py, το οποίο καλεί η main κατά την εκτέλεση της. Το output του query 3 βρίσκεται στο /output/output4.py
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.

```
    data3 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/LA_income_2015.csv", header=True, schema=schema2)
    data4 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/revgecoding.csv", header=True, schema=schema3)

    schema3 = "`LAT` DOUBLE, \
               `LON` DOUBLE, \
               `ZIPcode` INTEGER"

    schema2 = "`Zip Code` INTEGER, \
              `Community` STRING, \
              `Estimated Median Income` STRING"

    descent_mapping = {
        'A': 'Other Asian',
        'B': 'Black',
        'C': 'Chinese',
        'D': 'Cambodian',
        'F': 'Filipino',
        'G': 'Guamanian',
        'H': 'Hispanic/Latin/Mexican',
        'I': 'American Indian/Alaskan Native',
        'J': 'Japanese',
        'K': 'Korean',
        'L': 'Laotian',
        'O': 'Other',
        'P': 'Pacific Islander',
        'S': 'Samoan',
        'U': 'Hawaiian',
        'V': 'Vietnamese',
        'W': 'White',
        'X': 'Unknown',
        'Z': 'Asian Indian'
    }

    data3 = data3.withColumn("Estimated Median Income", regexp_replace(col("Estimated Median Income"), "\$", ""))
    data3 = data3.withColumn("Estimated Median Income", regexp_replace(col("Estimated Median Income"), ",", "").cast("
    float"))

    crime_year = df.withColumn("Year", year("DATE OCC"))

    crime_2015 = crime_year.filter(
       (col("Year") == 2015) &
       (col("Vict Descent").isNotNull()))

    def map_descent(code):
        return descent_mapping.get(code, "Unknown")  # Default to "Unknown" if code not found

    map_descent_udf = udf(map_descent, StringType())

    crime_2015 = crime_2015.withColumn("Vict Descent", map_descent_udf(crime_2015["Vict Descent"]))

    revgecoding = data4.dropDuplicates(['LAT', 'LON'])
```

Έπειτα υλοποιήθηκε το Query 3 χρησιμοποιώντας DataFrame/ SQL API.

```
    def query3 (crime_2015, data3, revgecoding):

        crime_zip = crime_2015.join(revgecoding, ["LAT", "LON"], "left")

        best3_zip = data3.orderBy("Estimated Median Income", ascending=False).limit(3)
```

```
        worst3_zip = data3.orderBy("Estimated Median Income", ascending=True).limit(3)

        best3_zip_list = [row['Zip Code'] for row in best3_zip.collect()]
        worst3_zip_list = [row['Zip Code'] for row in worst3_zip.collect()]

        crimes = crime_zip.filter(
            (col("ZIPcode").isin(best3_zip_list)) |
            (col("ZIPcode").isin(worst3_zip_list))
        )

        vict_descent_count = crimes.groupBy("Vict Descent").count().orderBy("count", ascending=False)


    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

Ο κώδικας για την εκτέλεση του query βρίσκεται στο /scripts/queries/query4.py το οποίο καλεί η main κατά την εκτέλεση της. Το output του query 4 βρίσκεται στο /output/output5.py
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.

```
    schema4 = "`X` DOUBLE, \
               `Y` DOUBLE, \
               `FID` INTEGER, \
               `DIVISION` STRING, \
               `LOCATION` STRING, \
               `PREC` INTEGER"

    data5 = spark.read.csv("/user/ubuntu/ta/advanced-db/data/LAPD_Police_Stations.csv", header=True, schema=schema4)
```

```
    def query4(df, data5):

        def haversine(lat1, lon1, lat2, lon2):
            # Radius of the Earth in kilometers
            R = 6371.0

            lat1_rad = math.radians(lat1)
            lon1_rad = math.radians(lon1)
            lat2_rad = math.radians(lat2)
            lon2_rad = math.radians(lon2)

            dlat = lat2_rad - lat1_rad
            dlon = lon2_rad - lon1_rad

            a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
            c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

            distance = R * c
            return distance

        def get_distance(lat1, long1, lat2, long2):

            def is_valid_coordinate(lat, lon):
                return -90 <= lat <= 90 and -180 <= lon <= 180


            if not is_valid_coordinate(lat1, long1) or not is_valid_coordinate(lat2, long2):
                # Print the invalid rows
                print(f"Invalid row: lat1={lat1}, long1={long1}, lat2={lat2}, long2={long2}")
                return -1

            try:
                return haversine(lat1, long1, lat2, long2)
            except ValueError:
                return -1

        df = df.filter(
            (df["AREA NAME"] != "Null Island") &
            (df["Weapon Used Cd"].substr(1, 1) == "1")
        )

        joined_df = df.join(data5, df["AREA"] == data5["PREC"])

        distance_udf = udf(get_distance)
```

```
distance_df = joined_df.withColumn(
    "DISTANCE",
    distance_udf(
        F.col("LAT"), F.col("LON"),
        F.col("Y"), F.col("X")
    ).cast("double")
)

query_4_1a = distance_df.groupBy("Year").agg(
    F.avg("DISTANCE").alias("average_distance"),
    F.count("*").alias("#")
).orderBy("Year").withColumnRenamed("Year", "year")

query_4_1b = distance_df.groupBy("DIVISION").agg(
    F.avg("DISTANCE").alias("average_distance"),
    F.count("*").alias("#")
).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")

print("Απόσταση από το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό:")
print("(a)")
query_4_1a.show()
print("(b)")
query_4_1b.show()

cross_joined_df = df.crossJoin(data5.withColumnRenamed("LAT", "Y").withColumnRenamed("LON", "X"))

cross_joined_df = cross_joined_df.withColumn(
    "DISTANCE",
    distance_udf(col("LAT"), col("LON"), col("Y"), col("X")).cast("double")
)

windowSpec = Window.partitionBy("DR_NO").orderBy("DISTANCE")

nearest_station_df = cross_joined_df.withColumn(
    "row_num",
    F.row_number().over(windowSpec)
).filter(col("row_num") == 1).drop("row_num")

query_4_2a = nearest_station_df.groupBy("Year").agg(
    F.avg("DISTANCE").alias("average_distance"),
    F.count("*").alias("#")
).orderBy("Year").withColumnRenamed("Year", "year")

query_4_2b = nearest_station_df.groupBy("DIVISION").agg(
    F.avg("DISTANCE").alias("average_distance"),
    F.count("*").alias("#")
).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")


print("Απόσταση από το πλησιέστερο αστυνομικό τμήμα:")
print("(a)")
query_4_2a.show()
print("(b)")
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:

(a)

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

(b)

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:

(a)

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

(b)

division	average_distance	#
77TH STREET	1.7215717802940704	13314
SOUTHWEST	2.281362128260118	11195
SOUTHEAST	2.2100092984159 25	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

Προσαρμώσαμε τους κώδικες για το Query 3 και το Query 4 ώστε να χρησιμοποιούν την τεχνική hint & explain η οποία μας εκτυπώνει το Plan που ακολουθείται για την εκτέλεση του join.
Αυτό βέβαια λόγω lazy evaluation δεν είναι final αφού το final plan είναι εκείνο που εκτελείται κατά τη διάρκει του .show(). Το οποίο βρίσκουμε στο UI του Spark.

Query 3

Για την εκτέλεση κάνουμε spark-submit /scripts/query3_join.py.
Το output βρίσκεται στο /output/output6.py.
Έκει βρίσκεται και το Plan που εκτυπώνει το .explain() το οποίο όμως δεν είναι το Final Plan.
Εκτελέσαμε το Query 3 σε ένα for loop που επιλέγει μια διαφορετική στρατηγική join σε κάθε επανάληψη.

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.

Link για το Spark UI: Query3 Hint & Explain

JOIN STRATEGY	DURATION
BROADCAST	35s
MERGE	21s
SHUFFLE HUSH	19s
SHUFFLE REPLICATE NL	28s

```python
join_strategies = ["broadcast", "merge", "shuffle_hash", "shuffle_replicate_nl"]

for strategy in join_strategies:
    print(f"Executing query with {strategy} join strategy")
    query3_hne_results = query3_hne.query3(crime_2015, data3, revgecoding, strategy)
    query3_hne_results.show()
```

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

```python
def query3(crime_2015, data3, revgecoding, join_strategy):

    crime_zip = crime_2015.join(revgecoding.hint(join_strategy), ["LAT", "LON"], "left")

    best3_zip = data3.orderBy("Estimated Median Income", ascending=False).limit(3)
    worst3_zip = data3.orderBy("Estimated Median Income", ascending=True).limit(3)

    best3_zip_list = [row['Zip Code'] for row in best3_zip.collect()]
    worst3_zip_list = [row['Zip Code'] for row in worst3_zip.collect()]

    crimes = crime_zip.filter(
        (col("ZIPcode").isin(best3_zip_list)) |
        (col("ZIPcode").isin(worst3_zip_list))
    )

    vict_descent_count = crimes.groupBy("Vict Descent").count().orderBy("count", ascending=False)

    vict_descent_count.explain()

    return vict_descent_count
```

Query 4

Για την εκτέλεση κάνουμε spark-submit /scripts/query4_broadcast.py, /scripts/query4_merge.py, /scripts/query4_shuffle_hush.py, /scripts/query4_shuffle_replicate_nl.py αντίστοιχα για κάθε join strategy.
Το output βρίσκεται από το /output/output7.py μέχρι το /output/output10.py.
Έκει βρίσκονται και τα Plans που εκτυπώνει το .explain() τα οποία όμως δεν είναι τα Final Plans.

Ο κώδικας για το Query 4 έχει συμπληρωθεί με τα απαραίτητα hint και explain. Τα αποτελέσματα της εκτέλεσης φαίνονται στους ακόλουθους συνδέσμους:
Query 4 Broadcast Query 4 Merge Query 4 Shuffle Hush Query 4 Shuffle Replicate

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.

(a)

JOIN STRATEGY	DURATION
BROADCAST	36s
MERGE	35s
SHUFFLE HUSH	36s
SHUFFLE REPLICATE NL	52s

(b)

JOIN STRATEGY	DURATION
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 κατανεμημένα 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.

```
    def query4(df, data5, join_strategy):

        def haversine(lat1, lon1, lat2, lon2):
            # Radius of the Earth in kilometers
            R = 6371.0

            lat1_rad = math.radians(lat1)
            lon1_rad = math.radians(lon1)
            lat2_rad = math.radians(lat2)
            lon2_rad = math.radians(lon2)

            dlat = lat2_rad - lat1_rad
            dlon = lon2_rad - lon1_rad

            a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
            c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

            distance = R * c
            return distance

        def get_distance(lat1, long1, lat2, long2):

            def is_valid_coordinate(lat, lon):
                return -90 <= lat <= 90 and -180 <= lon <= 180


            if not is_valid_coordinate(lat1, long1) or not is_valid_coordinate(lat2, long2):
                # Print the invalid rows
                print(f"Invalid row: lat1={lat1}, long1={long1}, lat2={lat2}, long2={long2}")
                return -1

```

```
        try:
            return haversine(lat1, long1, lat2, long2)
        except ValueError:
            return -1

    df = df.filter(
        (df["AREA NAME"] != "Null Island") &
        (df["Weapon Used Cd"].substr(1, 1) == "1")
    )

    joined_df = df.join(data5.hint(join_strategy), df["AREA"] == data5["PREC"])

    joined_df.explain()

    distance_udf = udf(get_distance)

    distance_df = joined_df.withColumn(
        "DISTANCE",
        distance_udf(
            F.col("LAT"), F.col("LON"),
            F.col("Y"), F.col("X")
        ).cast("double")
    )

    query_4_1a = distance_df.groupBy("Year").agg(
        F.avg("DISTANCE").alias("average_distance"),
        F.count("*").alias("#")
    ).orderBy("Year").withColumnRenamed("Year", "year")

    query_4_1b = distance_df.groupBy("DIVISION").agg(
        F.avg("DISTANCE").alias("average_distance"),
        F.count("*").alias("#")
    ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")

    print("Απόσταση από το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό:")
    print("(a)")
    query_4_1a.show()
    print("(b)")
    query_4_1b.show()

    cross_joined_df = df.crossJoin(data5.withColumnRenamed("LAT", "Y").withColumnRenamed("LON", "X"))

    cross_joined_df = cross_joined_df.withColumn(
        "DISTANCE",
        distance_udf(col("LAT"), col("LON"), col("Y"), col("X")).cast("double")
    )

    windowSpec = Window.partitionBy("DR_NO").orderBy("DISTANCE")

    nearest_station_df = cross_joined_df.withColumn(
        "row_num",
        F.row_number().over(windowSpec)
    ).filter(col("row_num") == 1).drop("row_num")

    query_4_2a = nearest_station_df.groupBy("Year").agg(
        F.avg("DISTANCE").alias("average_distance"),
        F.count("*").alias("#")
    ).orderBy("Year").withColumnRenamed("Year", "year")

    query_4_2b = nearest_station_df.groupBy("DIVISION").agg(
        F.avg("DISTANCE").alias("average_distance"),
        F.count("*").alias("#")
    ).orderBy(F.desc("#")).withColumnRenamed("DIVISION", "division")


    print("Απόσταση από το πλησιέστερο αστυνομικό τμήμα:")
    print("(a)")
    query_4_2a.show()
    print("(b)")
    query_4_2b.show()
```

Ο φάκελος scripts περιλαμβάνει όλους τους κώδικες που χρησιμοποιήθηκαν, ενώ ο φάκελος output όλα τα captured outputs κατά την διάρκεια εκτέλεσης των spark-submit.