



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ  
ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ  
ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΓΝΩΣΕΩΝ ΚΑΙ ΔΕΔΟΜΕΝΩΝ  
Ακαδημαϊκό έτος 2023-24, 9ο Εξάμηνο

**Εξαμηνιαία εργασία στα Προχωρημένα συστήματα Βάσεων Δεδομένων**

Ομάδα 24: Ευάγγελος Παπαμήτσος, Χρήστος Μεντζίνης

Η εργασία πραγματοποιήθηκε τοπικά σε έναν κόμβο λόγω μη λήψης πόρων στον ωκεανό και αποτυχίας δημιουργίας του κατάλληλου *distributed* περιβάλλοντος με VMs, docker...

Δημιουργούμε ένα DataFrame από το σύνολο δεδομένων και αφού κάνουμε τις ζητούμενες μετατροπές παίρνουμε τις παρακάτω πληροφορίες:

Total Rows: 2979324

-- DR_NO: integer (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)

Για το Query 1 θα χρησιμοποιήσουμε αρχικά το Dataframe API του Spark:

```
crime_data = Dataframe.withColumn("year", F.year("Date Rptd"))
crime_data = crime_data.withColumn("month", F.month("Date Rptd"))
count_df = crime_data.groupBy("year", "month").agg(F.count("*").alias("crime_count"))
window_spec = Window.partitionBy("year").orderBy(F.desc("crime_count"))
result = count_df.withColumn("ranking", F.row_number().over(window_spec)).filter("ranking <= 3") \
.orderBy("year", "ranking", "month")
```

Συγκεκριμένα, δημιουργούμε ένα νέο DataFrame από το αρχικό με δύο νέες στήλες month και year που δημιουργούμε με τις αντίστοιχες συναρτήσεις από την στήλη Date Rptd.

Στην συνέχεια με την μέθοδο groupBy αντιστοιχούμε κάθε ζεύγος year-month στον αντίστοιχο αριθμό εγκλημάτων που σημειώθηκαν και έπειτα δημιουργούμε ένα βοηθητικό παράθυρο στο οποίο θα δράσει η μέθοδος row\_number για να πάρουμε το ranking κάθε μήνα ανά χρόνο.

Τέλος, φιλτράρουμε για να κρατήσουμε τους 3 μήνες κάθε χρόνου με τα περισσότερα εγκλήματα και εμφανίζουμε τα αποτελέσματα με την ζητούμενη μορφή.

Την ίδια λογική θα χρησιμοποιήσουμε και με την χρήση του SQL API:

```
Dataframe.createOrReplaceTempView("Crime_data")
result = spark.sql("""
    WITH RankedCrimeData AS (
    SELECT
        YEAR(`Date Rptd`) AS year,
        MONTH(`Date Rptd`) AS month,
        COUNT(*) AS `crime_total`,
        ROW_NUMBER() OVER(PARTITION BY YEAR(`Date Rptd`) ORDER BY COUNT(*) DESC) as
`#`
    FROM Crime_data
    GROUP BY year, month
    )
    SELECT year, month, crime_total, `#`
```

```

FROM RankedCrimeData
WHERE `#` <= 3
ORDER BY year, `#`, month;
""")

```

Στην παρακάτω φωτογραφία βλέπουμε και τα αποτελέσματα τα οποία είναι προφανώς τα ίδια για τις δύο υλοποιήσεις χωρίς να παρατηρούμε διαφορά στην επίδοση.

year	month	crime_count	ranking
2010	3	17595	1
2010	7	17520	2
2010	5	17338	3
2011	8	17139	1
2011	5	17050	2
2011	3	16951	3
2012	8	17697	1
2012	10	17477	2
2012	5	17391	3
2013	8	17329	1
2013	7	16714	2
2013	5	16671	3
2014	10	14131	1
2014	7	14107	2
2014	9	13871	3
2015	8	18951	1
2015	10	18916	2
2015	7	18528	3
2016	8	19779	1
2016	10	19615	2
2016	7	19262	3
2017	10	20400	1
2017	8	20086	2
2017	7	19997	3
2018	5	20248	1
2018	7	19972	2
2018	10	19814	3
2019	7	19338	1
2019	8	19074	2
2019	3	18932	3
2020	1	18488	1
2020	2	17436	2
2020	7	17241	3
2021	10	23863	1
2021	11	23526	2
2021	9	23078	3
2022	5	21090	1
2022	1	20845	2
2022	8	20760	3
2023	8	20414	1
2023	1	20277	2
2023	7	20216	3

Για το Query 2 αρχικά θα χρησιμοποιήσουμε ξανά το Dataframe API του Spark:

```
street_crimes = Dataframe.filter(Dataframe['Premis Desc'] == 'STREET')
total_rows = street_crimes.count()
print(f'Total street crimes: {total_rows}')

street_crimes = street_crimes.withColumn(
    "Time of day",
    F.when((F.col("TIME OCC") >= 500) & (F.col("TIME OCC") < 1200), "morning")
    .when((F.col("TIME OCC") >= 1200) & (F.col("TIME OCC") < 1700), "noon")
    .when((F.col("TIME OCC") >= 1700) & (F.col("TIME OCC") < 2100), "afternoon")
    .when((F.col("TIME OCC") >= 2100) | (F.col("TIME OCC") < 500), "night")
    .otherwise("unknown")
)

street_crime_counts_by_time_of_day = street_crimes.groupBy("Time of
day").agg(F.count("*").alias("crime_count")).orderBy("crime_count")

street_crime_counts_by_time_of_day.show()
```

Αρχικά φιλτράρουμε το Dataframe για να εξετάσουμε μόνο τα εγκλήματα που έλαβαν χώρα στον δρόμο. Έπειτα προσθέτουμε μια νέα στήλη με την ονομασία της χρονοπεριόδου που έγινε το έγκλημα με βάση την στήλη TIME OCC. Τέλος, με την μέθοδο groupBy αντιστοιχούμε κάθε περίοδο με τον αντίστοιχο αριθμό εγκλημάτων και εμφανίζουμε το αποτέλεσμα.

Την ίδια διαδικασία θα ακολουθήσουμε χρησιμοποιώντας το RDD API:

```
Dataframe_rdd = Dataframe.rdd

street_crimes_rdd = Dataframe_rdd.filter(lambda row: row['Premis Desc'] == 'STREET')

def time_of_day(hour):
    if 500 <= hour < 1200:
        return "morning"
```

```

elif 1200 <= hour < 1700:
    return "noon"
elif 1700 <= hour < 2100:
    return "afternoon"
elif 2100 <= hour or hour < 500:
    return "night"
else:
    return "unknown"

```

```

street_crimes_rdd_with_time_of_day = street_crimes_rdd.map(lambda row: row + ("Time of day",
time_of_day(row['TIME OCC'])))

```

```

street_crime_counts_by_time_of_day_rdd = street_crimes_rdd_with_time_of_day.map(lambda row: (row[-
1], 1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1])

```

Συγκεκριμένα, αφού μετατρέψουμε το Dataframe σε κατάλληλη RDD μορφή, φιλτράρουμε πάλι κάθε row κρατώντας αυτά που αντιστοιχούν σε εγκλήματα που έγιναν στον δρόμο. Αφού ορίσουμε μια συνάρτηση που αντιστοιχίζει κατάλληλα τις τιμές της στήλης TIME OCC σε χρονοπεριόδους, εμπλουτίζουμε τις γραμμές με την νέα στήλη. Στην συνέχεια κάθε γραμμή αντιστοιχίζεται σε ένα ζευγάρι κλειδιού-τιμής, με κλειδί την χρονική περίοδο και η τιμή 1 και χρησιμοποιούμε την μέθοδο reduceByKey για μετρήσουμε τα αντίστοιχα εγκλήματα. Τέλος, ταξινομούμε το RDD σε αύξουσα σειρά.

Τα αποτελέσματα που προκύπτουν από τις δύο υλοποιήσεις είναι τα ίδια με την RDD υλοποίηση να υστερεί σε ταχύτητα.

```

+-----+-----+
|Time of day|crime_count|
+-----+-----+
|    morning|    123128|
|      noon|    147303|
|  afternoon|    186208|
|      night|    236350|
+-----+-----+

```

```

[('morning', 123128),
 ('noon', 147303),
 ('afternoon', 186208),
 ('night', 236350)]

```

Για το Query 3 θα χρησιμοποιήσουμε ξανά το Dataframe API του Spark:

```
crime_data_2015 = Dataframe.filter(F.year("Date Rptd") == 2015)
crime_data_2015 = crime_data_2015.filter(F.col("Vict Descent").isNotNull())
```

```
revgecoding_file = "./data/revgecoding.csv"
```

```
revgecoding_df = spark.read.csv(revgecoding_file, header=True, inferSchema=True)
```

Φιλτράρουμε το αρχικό Dataframe ώστε να κρατήσουμε τα εγκλήματα που έγιναν το 2015 και για τα οποία υπάρχει καταγεγραμμένη η καταγωγή του θύματος. Επίσης, δημιουργούμε ένα νέο Dataframe με βοηθητικές πληροφορίες για το zip code κάθε τοποθεσίας που μας ενδιαφέρει.

```
crime_data_2015 = crime_data_2015.join(revgecoding_df.hint(join_hint), ['LAT', 'LON'],
'left_outer').select("Vict Descent", "ZIPcode")
crime_data_2015.show()
```

Στην συνέχεια πραγματοποιούμε join των δύο Dataframes με βάση την τοποθεσία και κρατάμε τις στήλες που μας ενδιαφέρουν, δηλαδή την καταγωγή των θυμάτων και το zip code.

```
median_income_file = "./data/LA_income_2015.csv"
```

```
median_income_df = spark.read.csv(median_income_file, header=True, inferSchema=True)
```

```
median_income_df = median_income_df.withColumn("Estimated Median Income",
F.regexp_replace(F.col("Estimated Median Income"), "[^\\d.]", "").cast("int"))
```

```
top_incomes = median_income_df.orderBy(F.col("Estimated Median Income").desc()).limit(3)
```

```
lowest_incomes = median_income_df.orderBy(F.col("Estimated Median Income")).limit(3)
```

```
descents_low = crime_data_2015.join(lowest_incomes.hint(join_hint), F.col("ZIPcode") == F.col("Zip Code"), "inner")
```

```
descents_high = crime_data_2015.join(top_incomes.hint(join_hint), F.col("ZIPcode") == F.col("Zip Code"), "inner")
```

```
descents_low = descents_low.groupBy("Vict Descent").agg(F.count("*"))
```

```
descents_high = descents_high.groupBy("Vict Descent").agg(F.count("*"))
```

```
descents_low = descents_low.withColumnRenamed("Vict Descent", "Victim Descent")
```

```
descents_low = descents_low.withColumnRenamed("count(1)", "#")
```

```
descents_high = descents_high.withColumnRenamed("Vict Descent", "Victim Descent")
```

```
descents_high = descents_high.withColumnRenamed("count(1)", "#")
```

Αφού δημιουργήσουμε ένα νέο βοηθητικό Dataframe με πληροφορία για το μέσο εισόδημα και διαχωρίσουμε τις τρεις περιοχές με το υψηλότερο και χαμηλότερο εισόδημα κάνουμε join με βάση το zip code και αφού τα τροποποιήσουμε κατάλληλα εμφανίζουμε τα αποτελέσματα:

Victim Descent	#
O	102
H	51
A	16
W	312
X	26
B	14

Victim Descent	#
B	323
W	267
H	985
O	160
K	3
A	28
X	44
I	1
C	1

Για το Query 4 θα χρησιμοποιήσουμε πάλι το Dataframe API:

Αρχικά φιλτράρουμε το Dataset κρατώντας μόνο τα εγκλήματα με την χρήση πυροβόλου όπλου.

```
gun_crimes = Dataframe.filter(F.col('Weapon Used Cd').like('1%'))
```

Αφου δημιουργήσουμε ένα Dataframe με τις κατάλληλες πληροφορίες για την τοποθεσία των διάφορων τμημάτων του Los Angeles κάνουμε join τα δύο Dataframes και προσθέτουμε μία νέα στήλη με την απόσταση του εγκλήματος από το αστυνομικό τμήμα που το ανέλαβε.

```
gun_crimes = gun_crimes.withColumn("year", F.year("Date Rptd"))
gun_crimes = gun_crimes.join(LAPD_stations_df.hint(join_hint), F.col("AREA") ==
F.col("PREC"), "inner").select("year", "DIVISION", "LAT", "LON", "X", "Y")
gun_crimes = gun_crimes.withColumn("Distance", get_distance_udf("LON", "LAT", "X",
"Y")).select("year", "DIVISION", "Distance")
```

Τέλος, ομαδοποιούμε ανά έτος και υπολογίζουμε το πλήθος και την μέση απόσταση αλλά και ανά αστυνομικό τμήμα.

```
result = gun_crimes.groupBy("year").agg(
    F.avg("Distance").alias("avg_distance"),
    F.count("Distance").alias("#")
).orderBy("year")
result.show()

result = gun_crimes.groupBy("DIVISION").agg(
    F.avg("Distance").alias("avg_distance"),
    F.count("Distance").alias("#")
).orderBy(F.col("#").desc())
result.show()
```



year	avg_distance	#
2010	4.326341117149404	8162
2011	2.7904269556134778	7225
2012	37.48698144373706	6539
2013	2.830277105758418	5851
2014	10.470155426256689	4882
2015	2.7062554193223862	6729
2016	2.71765650940967	8094
2017	4.33923298743777	7781
2018	2.7355885132519364	7414
2019	2.74082683654595	7135
2020	8.613999633753169	8496
2021	32.849532153520755	12101
2022	2.612071114474666	10067
2023	2.557452978934544	8484

DIVISION	avg_distance	#
77TH STREET	5.745872330844248	16518
SOUTHEAST	13.56197627035242	13188
NEWTON	9.896845692177651	9581
SOUTHWEST	4.160551099530668	8615
HOLLENBECK	15.034085900271494	6095
HARBOR	13.377359089703047	5422
RAMPART	4.106286611009138	4976
MISSION	7.545645264792472	4450
OLYMPIC	1.8346273989145454	4305
NORTHEAST	10.459075190590411	3838
FOOTHILL	3.810820653031428	3761
HOLLYWOOD	12.119771738174023	3541
CENTRAL	4.789187936164047	3447
WILSHIRE	13.366513228684683	3420
NORTH HOLLYWOOD	14.03279776776525	3339
WEST VALLEY	17.119572432381975	2782
PACIFIC	13.276221563348491	2640
VAN NUYS	2.215864249480073	2637
DEVONSHIRE	18.571732831383127	2598
TOPANGA	3.4787633480390494	2300

Για το δεύτερο σκέλος του ερωτήματος η στήλη Distance υπολογίζεται αυτήν την φορά με μια συνάρτηση που υπολογίζει την απόσταση από το πλησιέστερο τμήμα.

```
gun_crimes = gun_crimes.withColumn("Distance", find_closest_police_station_udf("LON",
"LAT")).select("year", "DIVISION", "Distance")
```

Παρακάτω βλέπουμε τα νέα αποτελέσματα:

year	avg_distance	#
2010	3.9762380227744614	8162
2011	2.4582677442240546	7225
2012	37.13364209524676	6539
2013	2.4592839268144866	5851
2014	10.104120118078374	4882
2015	2.3883442547931346	6729
2016	2.425851163095081	8094
2017	4.00706930374014	7781
2018	2.411491332804097	7414
2019	2.4303624630677043	7135
2020	8.304682016178997	8496
2021	32.53589290418105	12101
2022	2.3151258839871804	10067
2023	2.2715967638620214	8484

DIVISION	avg_distance	#
77TH STREET	5.19309759370878	16518
SOUTHEAST	13.471602738284755	13188
NEWTON	9.444725167756307	9581
SOUTHWEST	3.5601573646826825	8615
HOLLENBECK	15.022619534245688	6095
HARBOR	13.290425819008904	5422
RAMPART	3.9680769025657825	4976
MISSION	6.768267212875467	4450
OLYMPIC	1.7393567696353007	4305
NORTHEAST	10.185143645838624	3838
FOOTHILL	3.7223003120431004	3761
HOLLYWOOD	12.112232296565534	3541
CENTRAL	4.715568664043192	3447
WILSHIRE	13.079051083443495	3420
NORTH HOLLYWOOD	13.83065202357141	3339
WEST VALLEY	16.481518315328803	2782
PACIFIC	13.190959733187475	2640
VAN NUYS	2.1875146795048224	2637
DEVONSHIRE	17.58786730446147	2598
TOPANGA	3.173372840027675	2300

Για τα joins των Queries 3, 4 χρησιμοποιήσαμε joins hints αν και έχουμε μόνο 1 node. Παρόλα αυτά θεωρούμε πως το broadcast hint είναι το κατάλληλο για τα περισσότερα.

Στο BROADCAST join το μικρότερο Dataframe γίνεται broadcast σε όλους τους κόμβους και είναι ιδιαίτερα αποδοτικό όταν αυτό είναι αρκετά μικρό για να χωρέσει στην μνήμη των κόμβων.

Στην περίπτωση μας τα joins πραγματοποιούνται με τα πολύ μικρότερα Dataframes LAPD\_stations\_df, top\_incomes, lowest\_incomes αλλά και σε τιμές (LAT, LON) που δεν είναι sorted για να χρησιμοποιήσουμε κάποιο MERGE.

Ολόκληρος ο κώδικας υπάρχει στο <https://github.com/VaggelisPap/AdvancedDB-Project>