Προχωρημένα Θέματα Βάσεων Δεδομένων

Πέτρος Ιάκωβος Φλωράτος (03121639)

Φίλιππος Γιαννακόπουλος (03121629)

GitHub Repository:

`https://github.com/FPG-992/AdvancedDatabases`

Query 1

> Να ταξινομηθούν, σε φθίνουσα σειρά, οι ηλικιακές ομάδες των θυμάτων σε περιστατικά που περιλαμβάνουν οποιαδήποτε μορφή "βαριάς σωματικής βλάβης". Θεωρείστε τις εξής ηλικιακές ομάδες: Παιδιά (<18), Νεαροί Ενήλικες(18-24), Ενήλικες (25-64), Ηλικιωμένοι (>64).

Ζητήθηκε να υλοποιηθεί το query χρησιμοποιώντας τα DataFrame (με και χωρίς UDF) και RDD API.

Σε αντίθεση με τις ενσωματωμένες συναρτήσεις του Spark οι οποίες μεταγλωτίζονται άμεσα σε βελτιστοποιημένο κώδικα Java, οι User Defined Functions επιτρέπουν την εκτέλεση custom Python συναρτήσεων που ορίζει ο χρήστης, τις οποίες όμως το Spark πρέπει να τις μετατρέψει σε κώδικα Java προκειμένου να τις χρησιμοπιήσει. Οπότε, περιμένουμε ότι η χρήση τους θα επιφέρει επιβαρύνσεις στην απόδοση λόγω της αναγκαίας διαδικασίας σειριοποίησης (serialization) για την μεταφορά δεδομένων μεταξύ της JVM και των Python workers.

Παρομοίως, στην προσέγγιση RDD (Resilient Distributed Dataset), τα δομημένα DataFrames μετατρέπονται σε αντικείμενα που επεξεργάζονται γραμμή προς γραμμή μέσω συναρτήσεων όπως map και reduce. Ως εκ τούτου, η προσέγγιση αυτή αναμένεται να είναι η πιο αργή, καθώς εκτός από το υψηλό κόστος σειριοποίησης (όπως και στα UDFs), τα RDDs αδυνατούν να αξιοποιήσουν τον Catalyst Optimizer για τη βελτιστοποίηση του πλάνου εκτέλεσης.

Για την εκτέλεση του ερωτήματος, και για τις τρεις προσεγγίσεις, χρησιμοποιήσαμε 4 executors με 1 πυρήνα και 2GB μνήμη για τον καθένα.

Τα πειραματικά αποτελέσματα ανέδειξαν πλήρως την θεωρητική ανάλυση, όπου η απλή υλοποίηση με Data Frames ήταν η ταχύτερη στα 9.89 δευτερόλεπτα, ακολουθούμενη από την υλοποίηση UDF στα 12.89 δευτερόλεπτα, με την προσέγγιση RDD να ήταν η πιο αργή, στα 14.64 δευτερόλεπτα.

Approach	**Execution Time (seconds)**
Native DataFrame	9.89
UDF DataFrame	12.89
RDD	14.64

Table 1: Χρόνοι εκτέλεσης ερωτήματος 1 για κάθε προσέγγιση

Η υπεροχή της native υλοποίησης πιστοποιεί στην πράξη την αποτελεσματικότητα της εκτέλεσης βελτιστοποιημένου bytecode εντός της JVM. Αντιθέτως, η αισθητή καθυστέρηση στην περίπτωση των UDFs αντανακλά το πρακτικό κόστος της "black box" διαχείρισης από τον Optimizer, ενώ η RDD υλοποίηση οριοθετεί το κάτω άκρο της απόδοσης, επιβαρυμένη από την απουσία των μηχανισμών βελτιστοποίηση, καθώς και από την αδυναμία μαζικής επεξεργασίας δεδομένων.

Age Group	Count
Adults (25–64)	121,660
Young Adults (18–24)	33,758
Children (<18)	10,904
Elderly (>64)	6,011

Table 2: Αποτελέσματα ερωτήματος 1: Κατανομή θυμάτων ανά ηλικιακή ομάδα

Query 2

Ανά έτος, να βρεθούν τα 3 φυλετικά γκρουπ με τα περισσότερα θύματα καταγεγραμμένων εγκλημάτων (Vict_Descent) στο Los Angeles. Τα αποτελέσματα να εμφανιστούν με φθίνουσα σειρά αριθμού θυμάτων ανά φυλετικό γκρουπ. Να υπολογιστεί και να εμφανιστεί επίσης το ποσοστό επί του συνολικού αριθμού θυμάτων ανα περίπτωση.

Ζητήθηκε να υλοποιηθεί το query χρησιμοποιώντας DataFrame και SQL APIs με στόχο τον εντοπισμό των κυριότερων δημογραφικών ομάδων θυμάτων ανά έτος. Και οι δύο υλοποιήσεος εκτελέστηκαν με 4 executors του 1 πυρήνα και 2ΓΒ μνήμης ο καθένας.

Η σύγκριση των χρόνων εκτέλεσης κατέδειξε αμελητέα διαφορά μεταξύ των δύο προσεγγίσεων, με την υλοποίηση DataFrame να ολοκληρώνεται σε 3.8065 δευτερόλεπτα και την SQL υλοποίηση σε 3.8769 δευτερόλεπτα. Το γεγονός ότι οι χρόνοι αυτοί είναι πρακτικά ταυτόσημοι επιβεβαιώνει την αποδοτικότητα του Catalyst Optimizer του Spark, ο οποίος μεταγλωττίζει και τις δύο συντακτικές μορφές στο ίδιο ακριβώς εσωτερικό λογικό και φυσικό πλάνο εκτέλεσης, αποδεικνύοντας ότι η επιλογή μεταξύ των δύο API αποτελεί ζήτημα προτίμησης κώδικα και αναγνωσιμότητας και όχι παράγοντα βελτιστοποίησης της απόδοσης.

Approach	Execution Time (seconds)
Native DataFrame	3.8065
SQL	3.8769

Table 3: Χρόνοι εκτέλεσης ερωτήματος 2 για κάθε προσέγγιση

Year	Victim Descent	Count (#)	Percentage (%)
2025	Hispanic/Latin/Mexican	34	40.48
2025	Unknown	24	28.57
2025	White	13	15.48
2024	Hispanic/Latin/Mexican	28,576	29.05
2024	White	22,958	23.34
2024	Unknown	19,984	20.32
2023	Hispanic/Latin/Mexican	69,401	34.55
2023	White	44,615	22.21
2023	Black	30,504	15.19
2022	Hispanic/Latin/Mexican	73,111	35.64
...	...	...	...

Table 4: Αποτελέσματα ερωτήματος 2: Κατανομή θυμάτων ανά έτος και φυλετική ομάδα

Query 3

> Να ταξινομηθούν και να εμφανιστούν με φθίνουσα σειρά συχνότητας εμφάνισης οι μέθοδοι διάπραξης εγκλημάτων και οι αντίστοιχοι κωδικοί τους (Mocodes). Χρησιμοποιήστε το σύνολο MO Codes για να αντιστοιχίσετε τους κωδικούς με τις περιγραφές τους.

Στο ερώτημα, αυτό, κληθήκαμε να χρησιμοποιήσουμε τόσο το DataFrame API όσο και το RDD API. Στην περίπτωση του DataFrame API ζητήθηκε να πειραματιστούμε με διάφορες μεθόδους joins, όπως Broadcast Hash Join, Merge Join, Shuffle Hash Join, και Cartesian Product. Για την πειραματική αξιολόγηση των διαφορετικών αλγορίθμων joins παρέκαμψαμε την προεπιλεγμένη απόφαση του Catalyst Optimizer κάνοντας χρήση της μεθόδου hint(). Μέσω αυτής, κάναμε tag τα DataFrames με metadata που υποδεικνύουν ρητά την επιθυμητή στρατηγική, εξαναγκάζοντας το Spark να προσαρμόσει ανάλογα το φυσικό πλάνο εκτέλεσης.

Το Broadcast Hash Join (BROADCAST) αποτελεί την προεπιλεγμένη και βέλτιστη στρατηγική όταν ενώνουμε έναν μεγάλο πίνακα με έναν πολύ μικρό πίνακα. Ο Driver node ανακτά ολόκληρο τον μικρό πίνακα στη μνήμη του και στη συνέχεια εκπέμπει (broadcasts) ένα αντίγραφο σε κάθε Executor. Έτσι, κάθε κόμβος εκτελεί το join τοπικά, αποφεύγοντας πλήρως το δαπανηρό shuffling του μεγάλου πίνακα. Προϋπόθεση για τη χρήση της είναι ο μικρός πίνακας να χωράει στη μνήμη, διαφορετικά προκαλείται σφάλμα OutOfMemory.

Το Sort Merge Join (MERGE) είναι η τυπική επιλογή για την ένωση δύο μεγάλων πινάκων. Αρχικά, εκτελείται Shuffle, μετακινώντας δεδομένα μέσω του δικτύου ώστε εγγραφές με ίδιο κλειδί να βρεθούν στον ίδιο κόμβο. Στη συνέχεια, και οι δύο πλευρές ταξινομούνται (Sort) βάσει του κλειδιού ένωσης και ακολουθεί μια διαδικασία συγχώνευσης (Merge) μέσω γραμμικής σάρωσης. Αν και εξαιρετικά ανθεκτική για μεγάλα δεδομένα, μειονεκτεί σε ταχύτητα λόγω του υψηλού κόστους I/O (κατά το shuffle) και CPU (κατά το sort).

Το Shuffled Hash Join (SHUFFLE_HASH) λειτουργεί παρόμοια με την Sort Merge ως προς την ανακατανομή των δεδομένων (Shuffle), αλλά παραλείπει τη φάση της ταξινόμησης. Αντ΄ αυτού, ο Executor δημιουργεί έναν πίνακα κατακερματισμού (Hash Map) στη μνήμη για το μικρότερο από τα δύο παρτιτιονς και στη συνέχεια σαρώνει το μεγαλύτερο παρτιτιον ελέγχοντας για αντιστοιχίες. Είναι αποδοτική για ενώσεις μεγάλων πινάκων με μεσαίου μεγέθους πίνακες.

Τέλος, το Cartesian Product (SHUFFLE_REPLICATE_NL) αντιστοιχεί στη στρατηγική Nested Loop. Συνήθως χρησιμοποιείται ως έσχατη λύση σε ενώσεις χωρίς κλειδιά ισότητας (non-equi joins). Απαιτεί την αντιγραφή τμημάτων του ενός πίνακα σε κάθε κόμβο που περιέχει τμήματα του άλλου, οδηγώντας σε εκρηκτική αύξηση του όγκου δεδομένων.

Αξίζει να σημειωθεί ότι κατά την εκτέλεση του πειράματος, ο Catalyst Optimizer αγνόησε το συγκεκριμένο hint για να προστατεύσει την εκτέλεση, επιλέγοντας αυτόματα την αποδοτικότερη BROADCAST στρατηγική.

Implementation	**Join Strategy (Hint)**	**Time (s)**
DataFrame API	Default (Broadcast)	11.67
DataFrame API	SHUFFLE_HASH	12.77
DataFrame API	MERGE	14.20
RDD API	-	17.47

Table 5: Σύγκριση χρόνων εκτέλεσης ερωτήματος 3 με διαφορετικές στρατηγικές

Συγκρίνοντας τις δύο κύριες υλοποιήσεις, η προσέγγιση μέσω DataFrame API (11.67 δευτερόλεπτα) αποδείχθηκε σημαντικά ταχύτερη από την αντίστοιχη RDD υλοποίηση (17.47 δευτερόλεπτα). Το αποτέλεσμα αυτό επιβεβαιώνει την υπεροχή του Catalyst Optimizer έναντι της δαπανηρής διαδικασίας σειριοποίησης αντικειμένων Python που επιβάλλει η χρήση των RDDs.

Μέσω της χρήσης των μεθόδων `ηιντ` και `εξπλαιν`, διαπιστώθηκε ότι η προεπιλεγμένη στρατηγική που επέλεξε ο Catalyst ήταν η Broadcast Hash Join, η οποία κατέγραψε και τον βέλτιστο χρόνο εκτέλεσης. Η συγκεκριμένη στρατηγική αξιολογείται ως η πλέον κατάλληλη για το συγκεκριμένο ερώτημα, καθώς ο πίνακας αναφοράς (MO Codes) διαθέτει εξαιρετικά μικρό μέγεθος.

Αντιθέτως, ο εξαναγκασμός χρήσης της στρατηγικής Shuffle Hash Join οδήγησε σε αύξηση του χρόνου εκτέλεσης στα 12.76 δευτερόλεπτα, εξαιτίας της απαίτησης για α-

νακατανομή δεδομένων μέσω του δικτύου. Ακόμη πιο αργή αποδείχθηκε η στρατηγική Sort Merge Join (14.20 δευτερόλεπτα), καθώς προσθέτει στο κόστος του shuffling και το υπολογιστικό κόστος της ταξινόμησης (sorting) και των δύο πινάκων.

MO Code	Description	Frequency
0344	Removes vict property	1,002,900
1822	Stranger	548,422
0416	Hit-Hit w/ weapon	404,773
0329	Vandalized	377,536
0913	Victim knew Suspect	278,618
2000	Domestic violence	256,188
1300	Vehicle involved	219,082
0400	Force used	213,165
1402	Evidence Booked (any crime)	177,470
1609	Smashed	131,229
...	...	...

Table 6: Αποτελέσματα ερωτήματος 3: Οι 10 συχνότερες μέθοδοι διάπραξης εγκλημάτων

```
== Physical Plan ==
AdaptiveSparkPlan (17)
+- Sort (16)
   +- Exchange (15)
      +- Project (14)
         +- BroadcastHashJoin LeftOuter BuildRight (13)
            :- HashAggregate (8)
            :  +- Exchange (7)
            :     +- HashAggregate (6)
            :        +- Filter (5)
            :           +- Generate (4)
            :              +- Project (3)
            :                 +- Filter (2)
            :                    +- Scan csv (1)
            +- BroadcastExchange (12)
               +- Project (11)
                  +- Filter (10)
                     +- Scan text (9)
```

Listing 1: Φυσικό Πλάνο Εκτέλεσης (Physical Plan) – Broadcast Hash Join

```
== Physical Plan ==
AdaptiveSparkPlan (19)
+- Sort (18)
   +- Exchange (17)
      +- Project (16)
         +- SortMergeJoin LeftOuter (15)
            :- Sort (9)
            :  +- HashAggregate (8)
            :     +- Exchange (7)
            :        +- HashAggregate (6)
            :           +- Filter (5)
            :              +- Generate (4)
            :                 +- Project (3)
            :                    +- Filter (2)
            :                       +- Scan csv  (1)
            +- Sort (14)
               +- Exchange (13)
                  +- Project (12)
                     +- Filter (11)
                        +- Scan text  (10)
```

Listing 2: Φυσικό Πλάνο Εκτέλεσης (Physical Plan) – Merge Sort Join

Query 4

> Να υπολογιστεί, ανά αστυνομικό τμήμα, ο αριθμός εγκλημάτων που έλαβαν χώρα πλησιέστερα σε αυτό από οποιοδήποτε άλλο, καθώς και η μέση απόστασή του από τις τοποθεσίες όπου σημειώθηκαν τα συγκεκριμένα περιστατικά. Τα αποτελέσματα να εμφανιστούν ταξινομημένα κατά αριθμό περιστατικών, με φθίνουσα σειρά

Για το ερώτημα, αυτό, χρησιμοποιήσαμε αποκλειστικά το DataFrame API και πειραματιστήκαμε με τον αριθμό των πυρήνων και της μνήμης κάθε executor (χρησιμοποιήσαμε σταθερά 2 executors για τα πειράματά μας).

Αναφορικά με τη στρατηγική ένωσης, το φυσικό πλάνο εκτέλεσης επιβεβαιώνει την ορθότητα της προηγούμενης ανάλυσης. Κεντρικό στοιχείο, όπως βλέπουμε πιο κάτω, αποτελεί ο τελεστής BroadcastNestedLoopJoin (BuildRight), ο οποίος αποδεικνύει ότι ο Catalyst Optimizer επέλεξε να διαμοιράσει τον μικρό πίνακα των αστυνομικών τμημάτων (Right side), αποφεύγοντας έτσι το δαπανηρό shuffling του ογκώδους συνόλου δεδομένων των εγκλημάτων. Η διαδικασία διαμοιρασμού διεκπεραιώνεται μέσω του τελεστή BroadcastExchange.

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [crime_id, crime_geom, DIVISION, station_geom, distance_m, (distance_m /
    1000.0) AS distance_km]
   +- Project [crime_id, crime_geom, DIVISION, station_geom, ST_DistanceSphere AS
       distance_m]
      +- BroadcastNestedLoopJoin BuildRight, Cross
         :- Project [crime_id, ST_Point AS crime_geom]
         :  +- Project [LAT, LON, monotonically_increasing_id() AS crime_id]
         :     +- Filter ((isnotnull(LAT) AND isnotnull(LON)) AND (NOT (LAT = 0.0) OR
             NOT (LON = 0.0)))
         :        +- FileScan csv [LAT,LON] ... PushedFilters: [IsNotNull(LAT),
             IsNotNull(LON), Or(Not(EqualTo(LAT,0.0)),Not(EqualTo(LON,0.0)))]
         +- BroadcastExchange IdentityBroadcastMode
            +- Project [DIVISION, ST_Point AS station_geom]
               +- FileScan csv [X,Y,DIVISION] ...
```

Listing 3: Φυσικό Πλάνο Εκτέλεσης (Physical Plan) – Query 4

Σχετικά με την επίδοση κατά την κλιμάκωση πόρων, παρατηρήθηκε σημαντική βελτίωση των χρόνων εκέλεσης με την αύξηση της διαθέσιμης υπολογιστικής ισχύος. Συγκεκριμένα, η διαμόρφωση με 1 πυρήνα και 2GB μνήμης ολοκλήρωσε την επεξεργασία σε 41.36 δευτερόλεπτα, ενώ η μετάβαση σε 2 πυρήνες και 4GB μνήμης μείωσε τον χρόνο στα 29.36 δευτερόλεπτα. Η βελτίωση αυτή αποδίδεται στην αποτελεσματικότερη παραλληλοποίηση των υπολογιστικά απαιτητικών συναρτήσεων υπολογισμού αποστάσεων (ST_DistanceSphere) και στην επάρκεια μνήμης που περιορίζει το overhead του Garbage Collection.

Τέλος, η περαιτέρω ενίσχυση σε 4 πυρήνες και 8GB μνήμης επέφερε επιπλέον μείωση στα 23.16 δευτερόλεπτα, επιδεικνύοντας συνεχιζόμενη βελτίωση, αν και με φθίνοντα ρυθμό, γεγονός που πιθανώς οφείλεται στα όρια που θέτει το I/O κατά την ανάγνωση των δεδομένων από τον δίσκο.

Division	**Avg. Distance (km)**	**Count (#)**
HOLLYWOOD	2.077	225,515
VAN NUYS	2.953	211,130
SOUTHWEST	2.191	189,565
WILSHIRE	2.593	187,061
77TH STREET	1.717	172,558
OLYMPIC	1.725	172,353
NORTH HOLLYWOOD	2.643	168,655
PACIFIC	3.853	162,514
...	...	...

Table 7: Αποτελέσματα ερωτήματος 4: Μέση απόσταση και πλήθος εγκλημάτων ανά αστυνομικό τμήμα

Cores (per Exec)	Memory (per Exec)	Time (s)
1	2 GB	41.36
2	4 GB	29.36
4	8 GB	23.16

Table 8: Χρόνοι εκτέλεσης ερωτήματος 4 με κλιμάκωση πόρων

Query 5

> Χρησιμοποιώντας ως αναφορά τα δεδομένα της απογραφής του 2020 για τον πληθυσμό και τα οικονομικά στοιχεία του 2021 για το εισόδημα ανα νοικοκυριό, να υπολογίσετε μέσα στη διετία 2020-2021 τη συσχέτιση μέσου ετήσιου κατακεφαλήν εισοδήματος με την ετήσια μέση αναλογία εγκλημάτων ανά άτομο για κάθε περιοχή του Λος Άντζελες. Επαναλάβετε τον υπολογισμό εξετάζοντας μόνο τις 10 περιοχές με το υψηλότερο και τις 10 με το χαμηλότερο ετήσιο κατακεφαλήν εισόδημα.

Για την υλοποίηση του πέμπτου ερωτήματος χρησιμοποιήθηκε το DataFrame API σε συνδυασμό με τη βιβλιοθήκη Apache Sedona για την εκτέλεση της χωρικής ένωσης (spatial join) μεταξύ των σημείων εγκλημάτων και των πολυγώνων των απογραφικών τετραγώνων.

Όσον αφορά τις στρατηγικές ένωσης που επέλεξε ο Catalyst Optimizer, παρατηρείται ένας συνδυασμός τεχνικών λόγω της πολυπλοκότητας του ερωτήματος. Αρχικά, για την ένωση των Census Blocks με τα εισοδηματικά δεδομένα (βάσει Zip Code), ο Optimizer επέλεξε την BroadcastHashJoin (βήμα 10 στο φυσικό πλάνο), καθώς ο πίνακας των εισοδημάτων είναι αρκετά μικρός ώστε να διαμοιραστεί σε όλους τους κόμβους, αποφεύγοντας το shuffling. Στη συνέχεια, για την κρίσιμη χωρική ένωση (RangeJoin στο βήμα 25) μεταξύ των εγκλημάτων και των περιοχών, ο Optimizer εφάρμοσε τη στρατηγική SortMergeJoin (βήμα 31) για την τελική συνένωση των αποτελεσμάτων. Η επιλογή αυτή κρίνεται λογική δεδομένου του μεγάλου όγκου δεδομένων και στις δύο πλευρές της ένωσης, καθιστώντας τη χρήση Broadcast απαγορευτική.

Σχετικά με τα αποτελέσματα της συσχέτισης, η ανάλυση ανέδειξε μια ασθενή αρνητική συσχέτιση ($r \approx -0.28$) στο σύνολο των περιοχών, υποδεικνύοντας ότι γενικά οι περιοχές με υψηλότερο εισόδημα τείνουν να παρουσιάζουν χαμηλότερα ποσοστά εγκληματικότητας. Εξετάζοντας τις ακραίες περιπτώσεις, στις 10 φτωχότερες περιοχές η αρνητική συσχέτιση ενισχύεται ($r \approx -0.40$), υποδηλώνοντας ότι στα χαμηλότερα εισοδηματικά στρώματα, ακόμα και μικρές οικονομικές διαφορές έχουν σημαντικό αντίκτυπο στην εγκληματικότητα. Αντίθετα, στις 10 πλουσιότερες περιοχές παρατηρήθηκε μια ασθενής θετική συσχέτιση

($r \approx 0.12$), φαινόμενο που υποδηλώνει ότι πέρα από ένα επίπεδο πλούτου, το εισόδημα παύει να λειτουργεί αποτρεπτικά.

```
== Physical Plan ==
AdaptiveSparkPlan (34)
+- Project (33)
   +- Project (32)
      +- SortMergeJoin LeftOuter (31)   <-- Final Aggregation Join
         :- Sort (16)
         :  +- Filter (15)
         :     +- HashAggregate (14)
         :        +- Exchange (13)
         :           +- HashAggregate (12)
         :              +- Project (11)
         :                 +- BroadcastHashJoin Inner BuildRight (10)  <-- Blocks +
             Income
         :                    :- Project (5)
         :                    :  +- Filter (4)
         :                    :     +- Generate (3)
         :                    :        +- Filter (2)
         :                    :           +- Scan json (1)
         :                    +- BroadcastExchange (9)
         :                       +- Project (8)
         :                          +- Filter (7)
         :                             +- Scan csv (6)
         +- Sort (30)
            +- HashAggregate (29)
               +- Exchange (28)
                  +- HashAggregate (27)
                     +- Project (26)
                        +- RangeJoin (25)   <-- Spatial Join (Crimes + Blocks)
                           :- Project (19)
                           :  +- Filter (18)
                           :     +- Scan csv (17)
                           +- Project (24)
                              +- Filter (23)
                                 +- Generate (22)
                                    +- Filter (21)
                                       +- Scan json (20)
```

Listing 4: Φυσικό Πλάνο Εκτέλεσης (Physical Plan) – Query 5

Dataset Subset	**Pearson Correlation (r)**
All Areas	-0.278
Top 10 Richest Areas	0.122
Bottom 10 Poorest Areas	-0.396

Table 9: Συσχέτιση Εισοδήματος και Εγκληματικότητας

Τέλος, όσον αφορά την απόδοση του συστήματος, παρατηρήθηκε ότι η αύξηση των πόρων δεν οδήγησε σε γραμμική μείωση του χρόνου εκτέλεσης (περίπου 108-110 δευτερόλεπτα σε όλες τις διαμορφώσεις). Το γεγονός αυτό υποδεικνύει ότι η διαδικασία κυριαρχείται (bottleneck) από το υπολογιστικό κόστος της χωρικής ένωσης (CPU bound λόγω γεωμετρικών υπολογισμών) και όχι από τη διαθέσιμη μνήμη ή το I/O.

Executors	Cores (per Exec)	Memory (per Exec)	Time (s)
2	4	8 GB	109.98
4	2	4 GB	108.16
8	1	2 GB	110.46

Table 10: Χρόνοι εκτέλεσης ερωτήματος 5 με κλιμάκωση πόρων

Community (COMM)	Population	Income PC ($)	Annual Crime Rate
Palisades Highlands	3,911	212,115	0.0139
Pacific Palisades	20,952	201,948	0.0302
Palos Verdes Peninsula	719	186,144	0.0000
Westfield/Academy Hills	1,396	186,144	0.0000
Bel Air	7,748	185,439	0.0330
Mandeville Canyon	3,242	163,295	0.0159
Beverly Crest	11,918	155,927	0.0258
Franklin Canyon	1	154,740	0.0000
Playa Vista	16,230	151,285	0.0308
Santa Monica Mountains	16,417	146,977	3.0045

Table 11: Αναλυτικά αποτελέσματα για τις 10 πλουσιότερες περιοχές (ερώτημα 5)