ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Επιστήμη Δεδομένων & Μηχανική Μάθηση

Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Ανάλυση Δεδομένων με χρήση Apache Hadoop και Apache Spark

Αναγνωσταράς Ιωάννης-Δημήτριος 03400242
Ρέντα Γεωργία-Άννα 03400270

Περιεχόμενα

1 Εισαγωγή

Η παρούσα εργασία έχει ως στόχο την ανάλυση μεγάλων συνόλων δεδομένων που αφορούν την εγκληματικότητα και τις δημογραφικές συνθήκες στην περιοχή του Los Angeles. Τα δεδομένα περιλαμβάνουν καταγραφές εγκλημάτων, δημογραφικά στοιχεία από την απογραφή του 2010, δεδομένα για το μέσο εισόδημα ανά νοικοκυριό και τις θέσεις των αστυνομικών τμημάτων της πόλης.Η ανάλυση αυτή έχει ως σκοπό την εξοικείωση με τη χρήση των κατανεμημένων συστημάτων Apache Spark και Apache Hadoop.

Τα scripts για κάθε ένα από τα ζητούμενα των ερωτημάτων που θα παρουσιαστούν, σχετικά αποτέλεσματα και εικόνες βρίσκονται στο αποθετήριο GitHub με τον ακόλουθο σύνδεσμο:
`https://github.com/Georgiannarenta/bigdata_dsml_project`

2 Query 1

Για το πρώτο ερώτημα ζητήθηκε να θεωρήσουμε τις ακόλουθες ηλικιακές ομάδες :

- Παιδιά:< 18
- Νεαροί ενήλικοι: 18–24
- Ενήλικοι: 25–64
- Ηλικιωμένοι: > 64

Έπειτα να ταξινομήσουμε τις ομάδες αυτές κατά φθίνουσα σειρά σε περιστατικά που περιλαμβάνουν οποιαδήποτε μορφή "βαριάς σωματικής βλάβης".Τα αποτελέσματα που πήραμε είναι τα ακόλουθα:

Ηλικιακή Ομάδα	**Περιστατικά**
Ενήλικοι	122727
Νεαροί ενήλικοι	34036
Παιδιά	16124
Ηλικιωμένοι	6082

Εξετάσαμε την απόδοση της υλοποίησης του Query 1 χρησιμοποιώντας τρεις διαφορετικές προσεγγίσεις: με RDD, με DataFrame χωρίς UDF και με DataFrame και UDF. Παρακάτω ακολουθεί η σύγκριση των τριών υλοποιήσεων με βάση τον χρόνο εκτέλεσης:

1. RDD

   Τα RDDs αποδείχθηκαν η πιο αργή προσέγγιση, με χρόνο εκτέλεσης 1.4 λεπτά καθώς δεν εκμεταλλεύονται τις βελτιστοποιήσεις του Catalyst και εκτελούνται ως δεδομένα σε

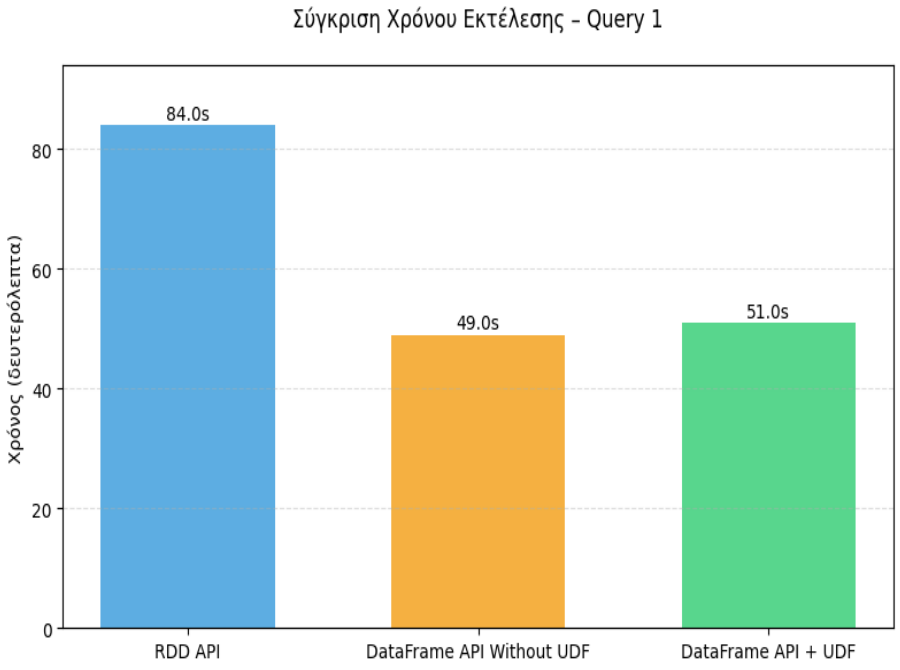
tuples, χωρις τη δυνατότητα ορισμού σχήματος που προσφέρουν τα DataFrames. Τα RDDs απαιτούν χειροκίνητο χειρισμό των δεδομένων, κάτι που καθιστά τον κώδικα πιο περίπλοκο και λιγότερο αποδοτικό.

2. DataFrame χωρίς UDF

   Η υλοποίηση του Query 1 χρησιμοποιώντας DataFrame χωρίς UDF αποδείχθηκε η ταχύτερη προσέγγιση, με συνολικό χρόνο εκτέλεσης 49 δευτερόλεπτα. Αυτή η προσέγγιση αξιοποιεί πλήρως τις βελτιστοποιήσεις του Catalyst Optimizer του Spark, που είναι υπεύθυνος για την αναγνώριση και εφαρμογή των βέλτιστων σχεδίων εκτέλεσης.

3. DataFrame με UDF
   Η εκτέλεση με χρήση User Defined Functions (UDFs) σε DataFrames ήταν πιο αποδοτική από τα RDDs, με χρόνο 51 δευτερόλεπτα. Ωστόσο, η χρήση UDF εισάγει επιπλέον κόστος, καθώς οι UDFs εκτελούνται εκτός του βελτιστοποιητή Catalyst, σε μη native περιβάλλον τύπου Python.

Η σύγκριση των τριών υλοποιήσεων παρουσιάζεται διαγραμματικά παρακάτω:



3 Query 2

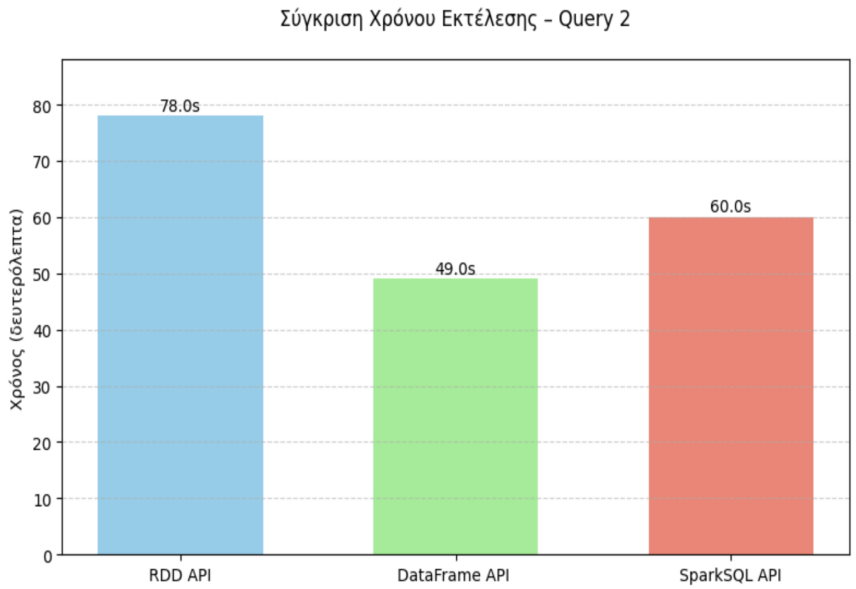
Το δεύτερο query ζητούσε την εύρεση των 3 τμημάτων με το υψηλότερο ποσοστό κλεισμένων υποθέσεων για κάθε έτος και την εμφάνιση του έτους,του ονόματος των τμημάτων καθώς και

τη θέση τους στην ετήσια κατάταξη.Ενδεικτικά κάποια από τα αποτελέσματα παραθέτονται παρακάτω:

Έτος	**Περιοχή (Precinct)**	**Closed Case Rate (%)**	**Κατάταξη**
2010	Rampart	32.947356	1
2010	Olympic	31.962706	2
2010	Harbor	29.632035	3
2011	Olympic	35.212168	1
2011	Rampart	32.511780	2
2011	Harbor	28.652205	3
2012	Olympic	34.414818	1
2012	Rampart	32.946418	2
2012	Harbor	29.815133	3

Top 3 περιοχές ανά έτος με βάση το ποσοστό επίλυσης υποθέσεων

Όσον αφορά τις υλοποιήσεις, όπως και στο πρώτο ερώτημα, η πιο αργή προσέγγιση ήταν αυτή με χρήση RDDs, με χρόνο εκτέλεσης 1.3 λεπτά. Αντίθετα, το SQL API του Spark επιτρέπει την εκτέλεση ερωτημάτων σε μορφή SQL πάνω σε δεδομένα που έχουν φορτωθεί ως πίνακες.Τόσο το DataFrame API όσο και το SQL API αξιοποιούν τον Catalyst Optimizer και βασίζονται στην ίδια λογική εκτέλεσης. Ωστόσο, στην πράξη παρατηρήθηκε ότι η υλοποίηση με SQL παρουσίασε ελαφρώς μεγαλύτερο χρόνο εκτέλεσης (1 λεπτό), σε σύγκριση με την αντίστοιχη με DataFrame API (49 δευτερόλεπτα). Αυτή η διαφορά οφείλεται κυρίως στο επιπλέον κόστος μετατροπής του SQL query σε λογικό πλάνο, καθώς και στη μειωμένη δυνατότητα ελέγχου της σειράς εκτέλεσης σε σχέση με τα DataFrames.



4 Query 3

Ζητούμενο του τρίτου query είναι ο υπολογισμός του μέσου ετησίου εισοδήματος ανά άτομο για κάθε ΤΚ του Los Angeles.
Κάποια από τα αποτελέσματα είναι τα ακόλουθα :

Zip Code	Income per Person ($)
91303	14867.86
91304	20446.33
91306	18758.84
91307	34390.03
91311	30298.12
91316	27229.64
91321	19588.49
91324	21002.71
91325	22343.38

Μέσο εισόδημα ανά άτομο ανά ταχυδρομικό κώδικα

Στην υλοποίηση του Query 3 με DataFrame API, δοκιμάσαμε την ανάγνωση των δεδομένων τόσο από αρχεία CSV όσο και από αρχεία Parquet. Το αποτέλεσμα ήταν ότι η εκτέλεση με αρχεία CSV είχε χρόνο 48 δευτερόλεπτα, ενώ με Parquet 47 δευτερόλεπτα. Γενικά, το Parquet είναι πιο αποδοτικό καθώς χρησιμοποιεί columnar μορφή αποθήκευσης, συμπιέζεται καλύτερα και διαβάζει μόνο τις απαραίτητες στήλες, κάτι που βοηθά σε μεγάλα σύνολα δεδομένων. Ωστόσο, στην περίπτωσή μας η διαφορά ήταν ελάχιστη, επειδή τα αρχεία είναι μικρά , οπότε δεν υπάρχει κάποιο σημαντικό πλεονέκτημα από τη χρήση του Parquet. Τέλος, όπως και στα προηγούμενα ερωτήματα, η υλοποίηση με χρήση RDD αποδείχθηκε η λιγότερο αποδοτική, με χρόνο εκτέλεσης 2.6 λεπτά.

5 Query 4

Στο τέταρτο query στόχος μας ήταν ο υπολογισμός του αριθμού εγκλημάτων ανά αστυνομικό τμήμα που έγιναν πλησιέστερα σε αυτό και περιλάμβαναν όπλα σε φθίνουσα σειρά καθώς και της μέσης απόστασης αυτού από τις τοποθεσίες των συμβάντων. Ορισμένα από τα αποτελέσματα είναι :

Division	Total Incidents	Average Distance
77TH STREET	52135	2.628
SOUTHWEST	40199	2.595
SOUTHEAST	37887	2.073
CENTRAL	36000	0.969
NEWTON	33053	2.065
RAMPART	31538	1.522
OLYMPIC	28671	1.756
HOLLYWOOD	24405	1.392
MISSION	23207	4.713
HARBOR	23204	3.902
HOLLENBECK	20478	2.561
FOOTHILL	19167	4.154

Αριθμός εγκλημάτων με χρήση όπλου και μέση απόσταση περιστατικών από το πλησιέστερο αστυνομικό τμήμα ανά τμήμα

Για την αξιολόγηση απόδοσης της υλοποίησης, εφαρμόσαμε διαφορετικές στρατηγικές κλιμάκωσης πόρων χρησιμοποιώντας τις διάφορες ρυθμίσεις για τον αριθμό των executors,των πυρήνων και της μνήμης ανά πυρήνα.

1. Horizontal Scaling : Στην οριζόντια κλιμάκωση αυξήσαμε τον αριθμό των executors διατηρώντας σταθερό το συνολικό πλήθος των πυρήνων (8 cores) και της μνήμης (16 GB). Συγκεκριμένα, δοκιμάσαμε τα εξής configurations:

   - 2 executors × 4 cores / 8 GB μνήμη
   - 4 executors × 2 cores / 4 GB μνήμη
   - 8 executors × 1 core / 2 GB μνήμη

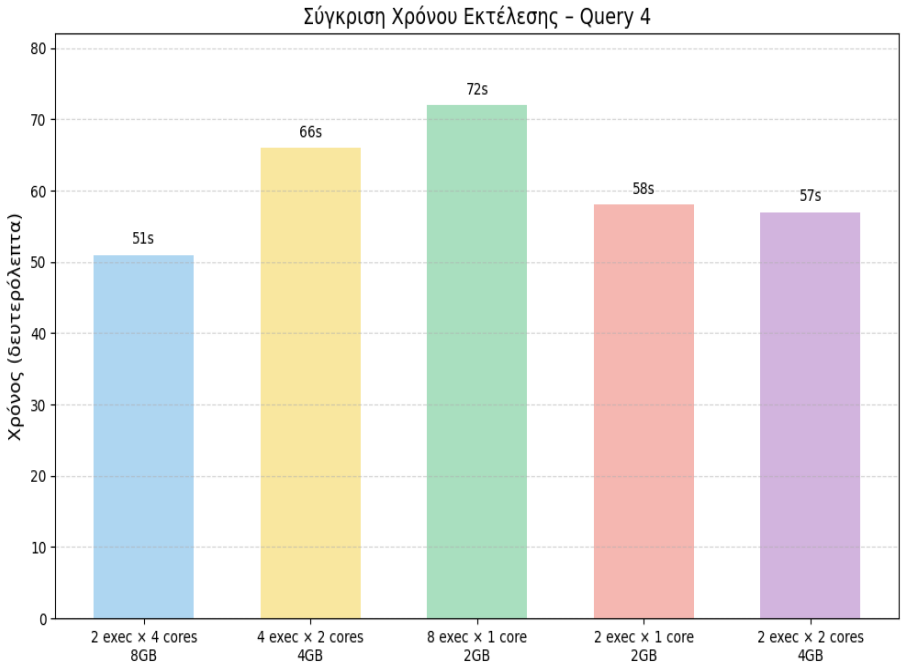
   Τα αποτελέσματα έδειξαν ότι η εκτέλεση ήταν πιο γρήγορη στους 2 executors με 4 cores, ενώ η περαιτέρω αύξηση του αριθμού των executors σε 8 επιδείνωσε τον χρόνο εκτέλεσης, λόγω αυξημένου κόστους επικοινωνίας και συντονισμού μεταξύ πολλών εκτελεστών. Αντίστοιχα, οι 4 executors με λιγότερους πυρήνες και μνήμη ανά executor παρουσίασαν επίσης χειρότερη απόδοση,λόγω περιορισμένων υπολογιστικών πόρων που δεν επέτρεψαν αποτελεσματική παραλληλοποίηση.

2. Vertical Scaling : Στο vertical scaling μεταβάλλαμε τους πόρους ανά executor, κρατώντας σταθερό τον αριθμό executors:

   - 2 executors × 1 core / 2 GB μνήμη
   - 2 executors × 2 cores / 4 GB μνήμη
   - 2 executors × 4 cores / 8 GB μνήμη

Παρατηρήθηκε μείωση του χρόνου εκτέλεσης όσο αυξάνονταν τα cores και η μνήμη ανά executor, κάτι που οφείλεται στην καλύτερη χρήση των πόρων από κάθε executor και την ταχύτερη επεξεργασία των δεδομένων.

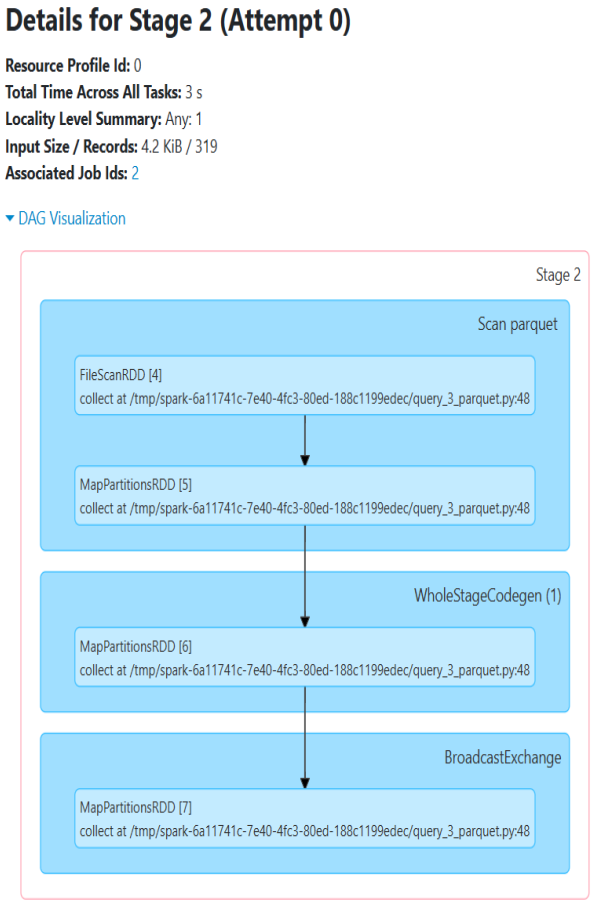
Η καλύτερη απόδοση επιτεύχθηκε με 2 executors × 4 πυρήνες / 8 GB μνήμη, που συνδυάζει ικανοποιητική παραλληλοποίηση και ισχυρούς πόρους ανά executor. Η πολύ μεγάλη κλιμάκωση με 8 executors × 1 πυρήνα δεν βελτίωσε τον χρόνο λόγω της διαχείρισης πολλών executors. Αντίθετα, η πολύ περιορισμένη κάθετη κλιμάκωση 2 executors × 1 πυρήνα υστερούσε λόγω έλλειψης παραλληλισμού. Η σύγκριση των υλοποιήσεων παρουσιάζεται διαγραμματικά παρακάτω:



6 Catalyst Optimizer και επιλογή Join στρατηγικής

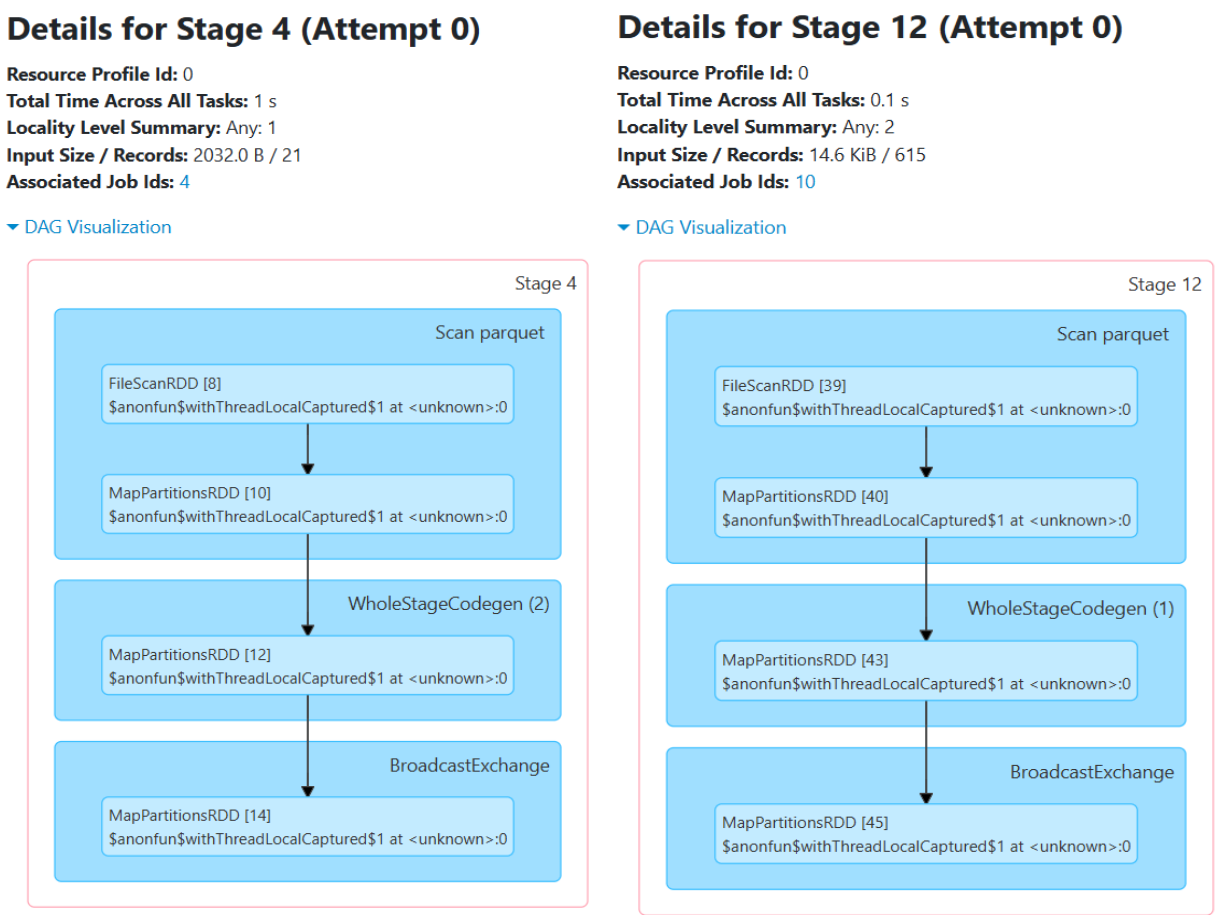
Κατά την εκτέλεση των Query 3 και Query 4, παρατηρήθηκε ότι ο Catalyst Optimizer επέλεξε τη στρατηγική Broadcast Join για όλα τα joins.
Το Broadcast Join είναι μία στρατηγική εκτέλεσης join που εφαρμόζεται όταν ένας από τους δύο πίνακες του join είναι σημαντικά μικρότερος από τον άλλο .Αντί να μετακινούνται και τα δύο σύνολα δεδομένων μέσω του δικτύου, ο μικρός πίνακας μεταδίδεται σε όλους τους executors, ώστε να αποφευχθεί η δαπανηρή ανακατανομή των δεδομένων.Αυτή η προσέγγιση μειώνει δραστικά την επιβάρυνση του δικτύου και επιτρέπει την τοπική εκτέλεση του join, προσφέροντας ταχύτερη και πιο αποδοτική επίδοση.Η επιλογή αυτή αιτιολογείται από τα μεγέθη των datasets.
Συγκεκριμένα το Query 3 απαιτούσε τη συνένωση δύο μικρών πινάκων, του 2010_Census_Populations_by_Zip _Code με 319 και του LA_income_2015 με 284 γραμμές.Λόγω του μικρού μεγέθους

και των δύο, ο Optimizer επέλεξε Broadcast Join, καθώς τα δεδομένα μπορούν να μεταδοθούν γρήγορα και να αποθηκευτούν τοπικά στη μνήμη.



Επιλογή join στρατηγικής για τη συνένωση του Query 3

Όσον αφορά το Query 4, η ένωση των δύο πινάκων με δεδομένα εγκλημάτων από το 2010 έως το 2025 σε έναν ενιαίο πίνακα με 3,138,316 γραμμές αποτελεί το βασικό fact table και δεν μεταδίδεται.Όλοι οι πίνακες που ενώνονται μαζί του(MO_codes, LA_Police_Stations με 830 και 21 γραμμές αντίστοιχα) είναι πολύ μικροί, άρα μεταδίδονται με broadcast χωρίς επιβάρυνση στο δίκτυο.



Επιλογή join στρατηγικής για τις δύο συνενώσεις του Query 4 (Crime_Data με MO_codes και έπειτα με LA_Police_Stations)