

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Προχωρημένα Θέματα Βάσεων Δεδομένων

Κυριακή Καρατζούνη el20634

Βικέντιος Βιτάλης el18803

Ομάδα 17

Αναφορά

Github repository: <https://github.com/VikentiosVitalis/advanced_topics_in_database_systems>

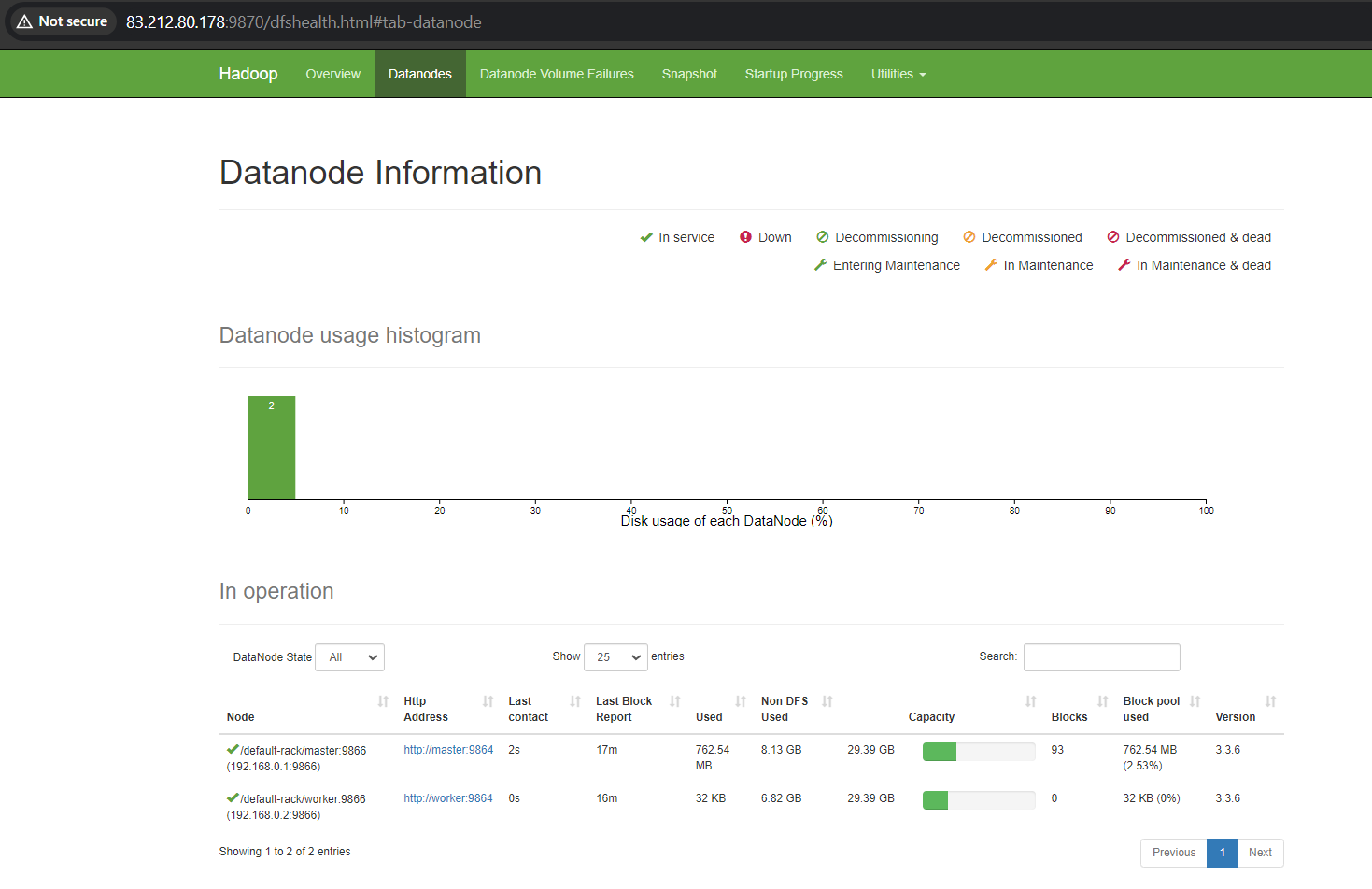
**Ζητούμενο 1.** Αρχικά δημιουργήσαμε στην υπηρεσία Okeanos Knossos ένα δίκτυο (cluster) 2 κόμβων σύμφωνα με τον εργαστηριακό οδηγό Advanced Topics in Database Systems: Lab guide.ipynb και εγκαταστήσαμε το λογισμικό και στους δύο κόμβους.

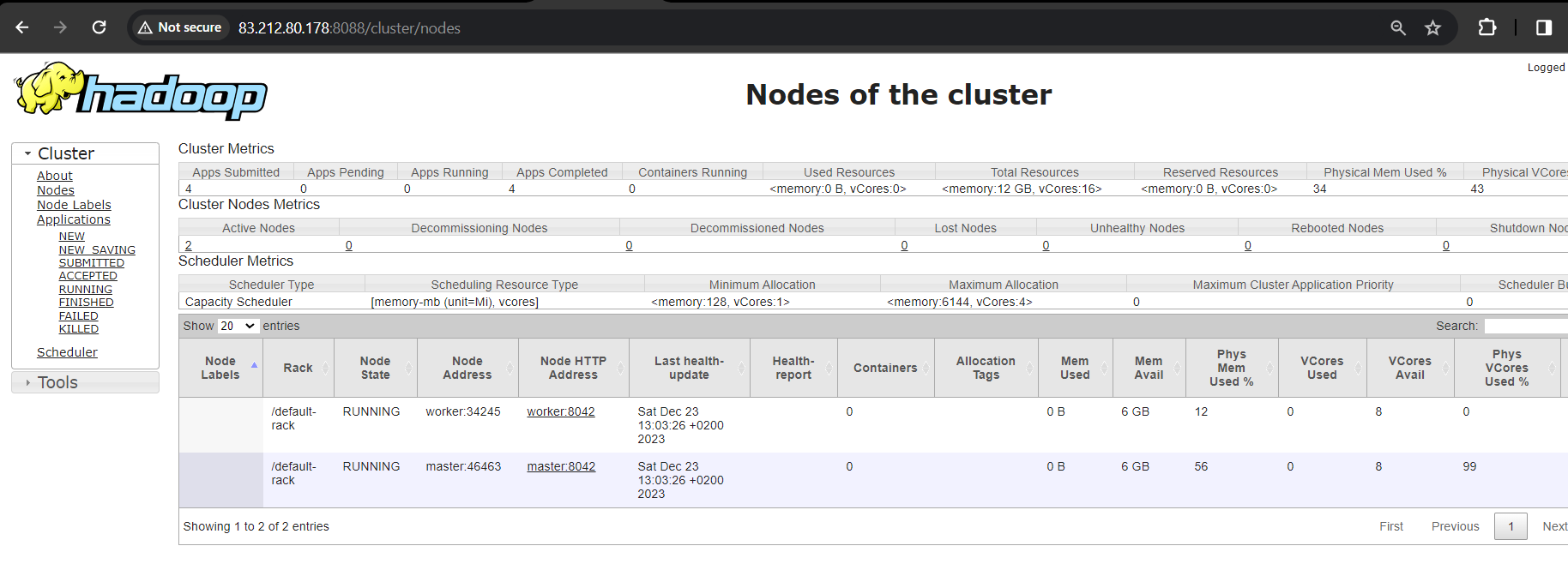
Μέσω του WinSCP συνδεθήκαμε στον master node κι από την επιφάνεια εργασίας των Windows μεταφορτώσαμε τα σύνολα δεδομένων. Παρατίθονται τα UIs από τις υπηρεσίες HDFS, YARN και Spark History Server αντίστοιχα:

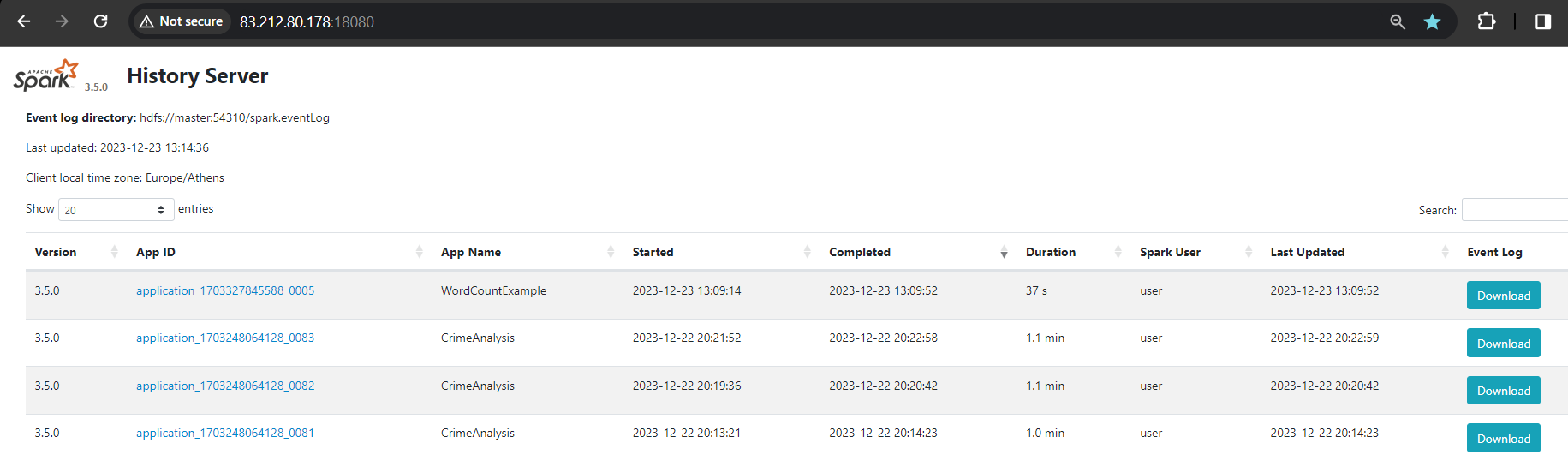
<http://83.212.80.178:9870/dfshealth.html#tab-datanode>

<http://83.212.80.178:8088/cluster/nodes>

<http://83.212.80.178:18080/>

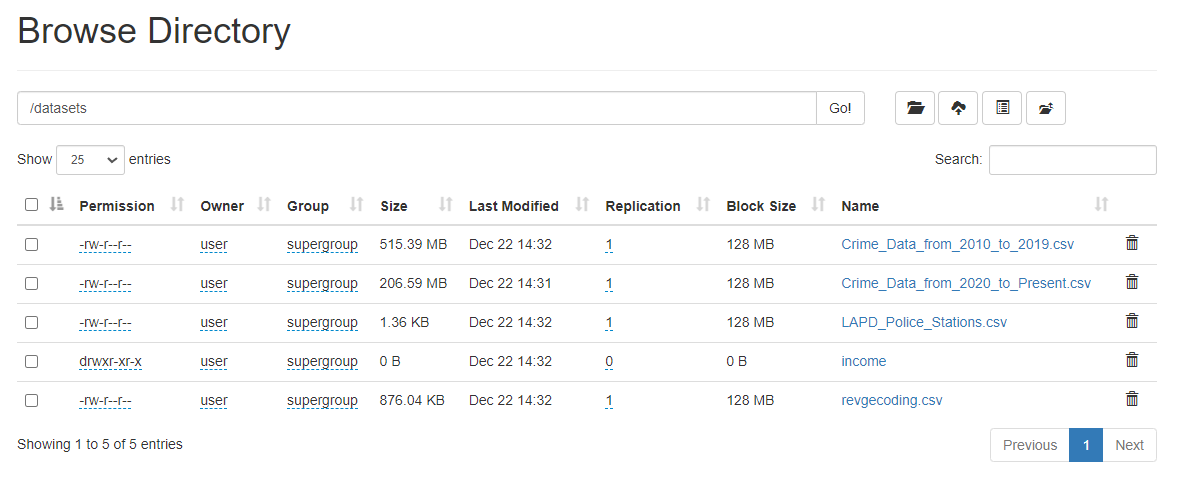




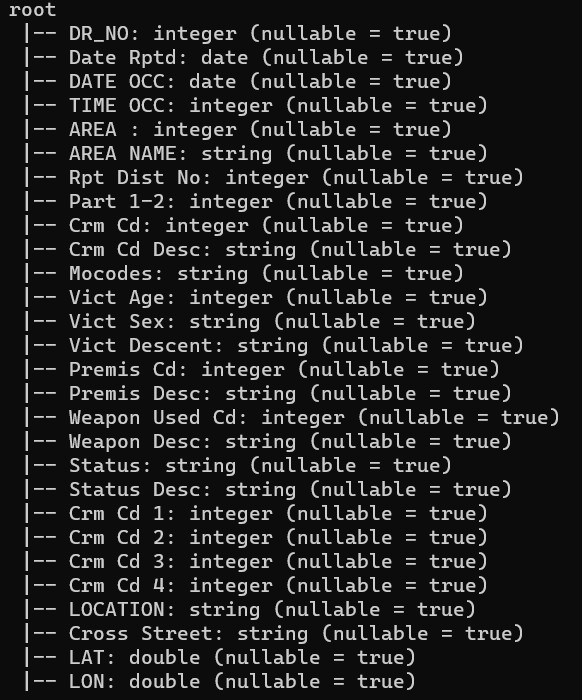


Με τις παρακάτω εντολές μεταφορτώνουμε τα αρχεία μας στην HDFS υπηρεσία που είναι διαθέσιμα στα Utilities > Browse the file system

* hadoop fs -mkdir hdfs://master:54310/datasets
* hadoop fs -mkdir hdfs://master:54310/datasets/income
* hadoop fs -put datasets/Crime\_Data\_from\_2010\_to\_2019.csv hdfs://master:54310/datasets/.
* hadoop fs -put datasets/Crime\_Data\_from\_2020\_to\_Present.csv hdfs://master:54310/datasets/.
* hadoop fs -put datasets/revgecoding.csv hdfs://master:54310/datasets/.
* hadoop fs -put datasets/LAPD\_Police\_Stations.csv hdfs://master:54310/datasets/.
* hadoop fs -put datasets/income/LA\_income\_2015.csv hdfs://master:54310/datasets/income/.
* hadoop fs -put datasets/income/LA\_income\_2017.csv hdfs://master:54310/datasets/income/.
* hadoop fs -put datasets/income/LA\_income\_2019.csv hdfs://master:54310/datasets/income/.
* hadoop fs -put datasets/income/LA\_income\_2021.csv hdfs://master:54310/datasets/income/.

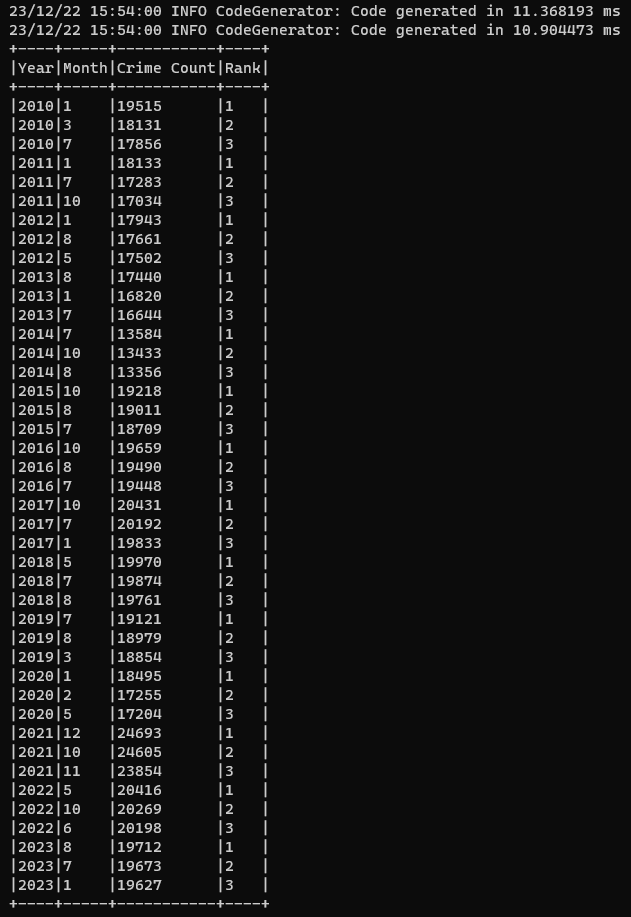


**Ζητούμενο 2.** Στο αρχείο queries/dataframe.py του github repository βρίσκεται η υλοποίηση του ερωτήματος σε python. Αρχικά αρικοποιούμε το spak seassion, διαβάζουμε το αρχείο Crime\_Data\_from\_2010\_to\_2019.csv, μετατρέπουμε τις στήλες στους αντίστοιχους τύπους δεδομένων, τυπώνουμε τους τύπους δεδομένων κάθε στήλης και τις συνολικές γραμμές .Παρακάτω φαίνεται η έξοδος στο Apache Spark περιβάλλον μετά την εκτέλεση του script μέσω spark-submit dataframe.py.

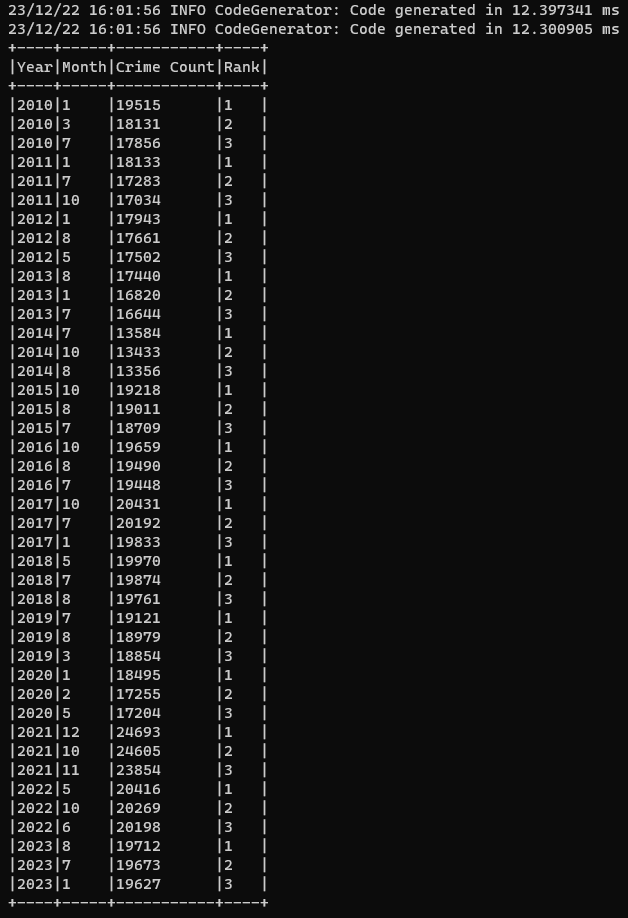




**Ζητούμενο 3.** Προκειμένου να υλοποιήσουμε το Query 1 με χρήση DataFrame, δημιουργούμε μια περίοδο Spark με 4 executors, δεικτοδοτούμε τα file paths με στόχο την ενοποίηση των δεδομένων εγκλημάτων για όλα τα διαθέσιμα έτη. Μετατρέπουμε την στήλη ‘DATE OCC’ σε τύπο δεδομένου datetime και εξάξουμε χρόνο και μήνα. Ομαδοποιούμε με βάση το πλήθος των εγκλημάτων και βρίσκουμε τους 3 μήνες με τα περισσότερα εγκλήματα για κάθε χρονιά. Τυπώνουμε τα αποτελέσματα και σταματάμε την περίοδο Spark. Εκτελούμε το Query μέσω της εντολής spark-submit q1df.py. Παρακάτω φαίνεται το αποτέλεσμα του Query1 χρησιμοποιώντας DataFrame μέσα από το Apache Spark περιβάλλον.



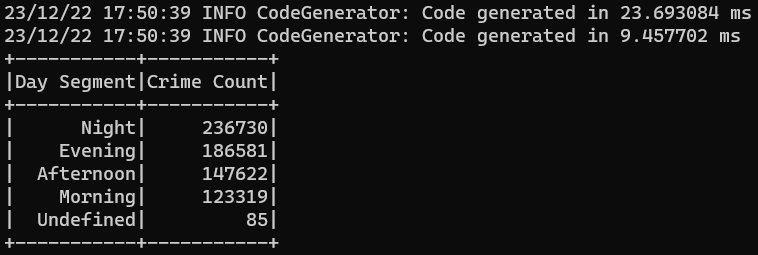
Εν συνεχεία, έχουμε το Query 1 με χρήση SQL API. Δημιουργούμε περίοδο Spark με 4 executors, τοποθετούμε τα .csv αρχεία σε Spark Data Frames, τα ενωποιούμε, μετατρέπουμε την στήλη ‘DATE OCC’ σε τύπο δεδομένου datetime, καταχωρούμε το Data Frame σε ένα προσωρινό SQL view, γράφουμε το SQL Query για την επεξεργασία δεδομένων, το εκτελούμε, τυπώνουμε το αποτέλεσμα και σταματάμε την περίοδο Spark. Παρακάτω φαίνεται το αποτέλεσμα του Query 1 χρησιμοποιώντας SQL API, μέσα από το Apache Spark περιβάλλον.



Παρατηρώντας τους χρόνους εκτέλεσης, συμπεραίνουμε ότι οι υλοποίησεις DataFrame API και SQL API είναι πολύ κοντινές από άποψη αποδοτικότητας, με την DataFrame API να πετυχαίνει ελαφρώς καλύτερο χρόνο. Αυτό συμβαίνει διότι στην περίπτωση μας το σύνολο των δεδομένων επεξεργάζεται σχεδόν εφάμιλλα κι από τα δυο APIs.

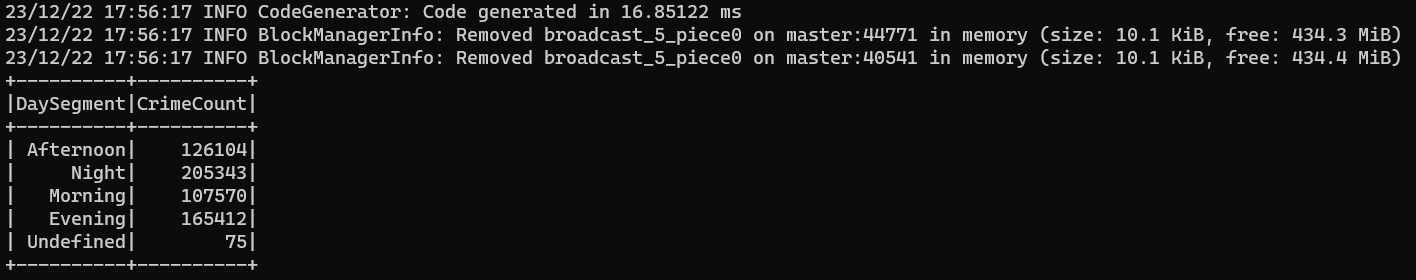
**Ζητούμενο 4.** Για την υλοποίηση του Query 2 χρησιμοποιώντας DataFrame δημιουργούμε μια περίοδο Spark, φτιάχνουμε μια συνάρτηση κατηγοριοποίησης της ημέρας σε Πρωϊ, Μεσημέρι, Απόγευμα και Βράδυ, καταχωρούμε την συνάρτηση που φτιάξαμε ως ορισμένη από τον χρήστη, διαβάζουμε τα αρχεία και τα ενωποιούμε, τα τοποθετούμε σε Data Frames, εφαρμόζουμε την συνάρτηση classify\_time\_segment

Παρακάτω φαίνεται η υλοποίηση του Query2 χρησιμοποιώντας DataFrame.

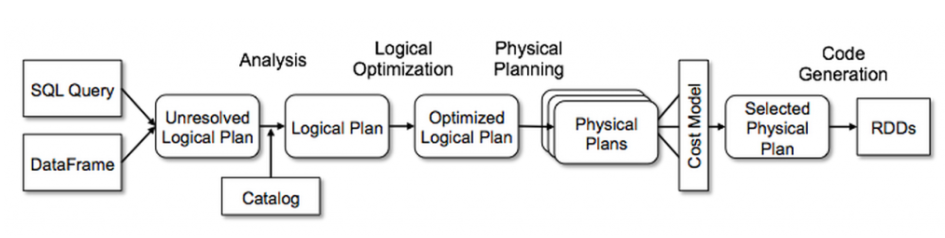


Εν συνεχεία, έχουμε το Query 2 με χρήση RDD API. Δημιουργούμε περίοδο Spark, φτιάχνουμε μια συνάρτηση κατηγοριοποίησης της ημέρας σε Πρωϊ, Μεσημέρι, Απόγευμα και Βράδυ, διαβάζουμε τα αρχεία σε RDDs και τα ενωποιούμε, χωρίζουμε κάθε γραμμή σε στήλες, εφαρμόζουμε τη συνάρτηση και φιλτράρουμε τα εγκλήματα δρόμου.

Παρακάτω φαίνεται η υλοποίηση του Query2 χρησιμοποιώντας RDD API.

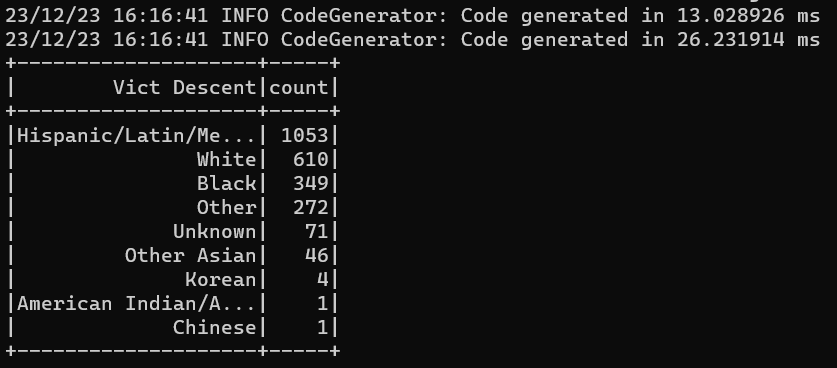


Παρατηρώντας τους χρόνους εκτέλεσης, όταν χρησιμοποιούμε τα DataFrame, ο κώδικας ειναι πιο αποδοτικός. Αυτό συμβαίνει επειδή τα DataFrames στο Spark είναι χτισμένα πάνω στη μηχανή Spark SQL, η οποία χρησιμοποιεί τον βελτιστοποιητή Catalyst. Επιπλέον, τα DataFrames βελτιστοποιούν τη χρήση μνήμης για δομημένα δεδομένα σε σύγκριση με τα RDDs. Αυτό έχει ως αποτέλεσμα καλύτερες επιδόσεις, ειδικά για μεγάλα σύνολα δεδομένων.

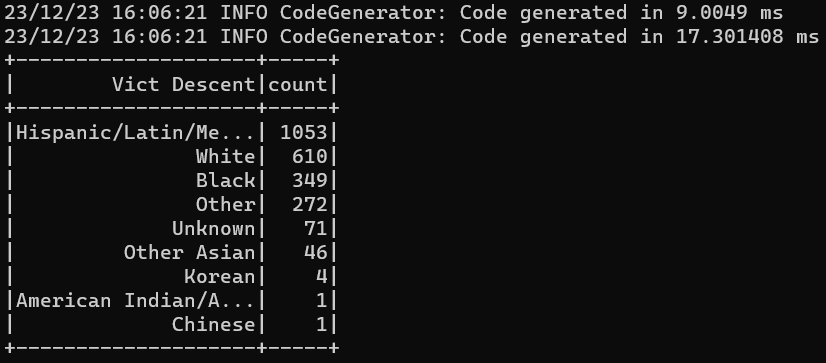


**Ζητούμενο 5.** Για την υλοποίηση του Query 3 δημιουργούμε περίοδο Spark με 2 executors, φορτώνουμε και διαβάζουμε τα αρχεία, μετατρέπουμε τα δεδομένα εισοδήματος σε αριθμητικά αφού αφαιρέσουμε το σύμβολο του δολαρίου και τα κόμματα, φιλτράρουμε μόνο τα δεδομένα για το 2015 και αποκλείουμε τις περιπτώσεις χωρίς καταγωγή θύματος. Έπειτα, κάνουμε MAP τα LAT και LON σε ZIP Codes, εντοπίζουμε τα 3 ZIP Codes με το υψηλότερο και χαμηλότερο εισόδημα και δημιουργούμε μια συνάρτηση, ώστε να κάνουμε την αντιστοίχηση των γραμμάτων με τις καταγωγές. Εμφανίζουμε τα αποτελέσματα και επαναλαμβάνουμε την ίδια διαδικασία χρησιμοποιώντας 3 και 4 executors.

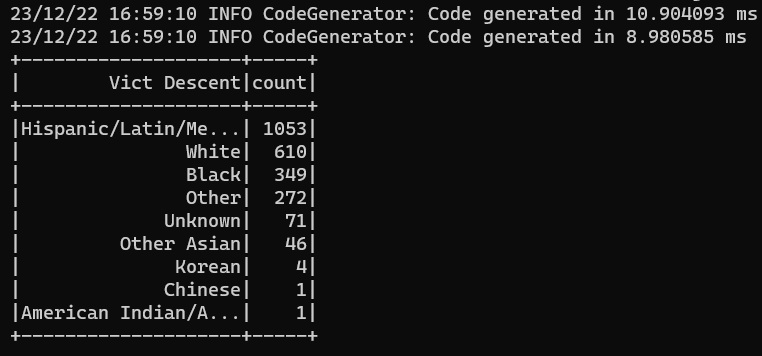
Παρακάτω φαίνεται η έξοδος κι ο χρόνος του Query3 χρησιμοποιώντας 2 Spark executors.



Παρακάτω φαίνεται η έξοδος κι ο χρόνος του Query3 χρησιμοποιώντας 3 Spark executors.



Παρακάτω φαίνεται η η έξοδος κι ο χρόνος του Query3 χρησιμοποιώντας 4 Spark executors.

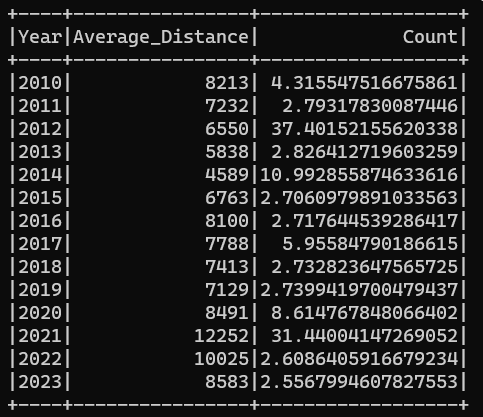


Παρατηρούμε ότι, αφού όσο περισσότερους εκτελεστές έχουμε, τόσο περισσότερες εργασίες μπορούν να εκτελούνται παράλληλα, η υλοποίηση με 4 executors είναι πιο αποδοτική και γρηγορότερη.

s

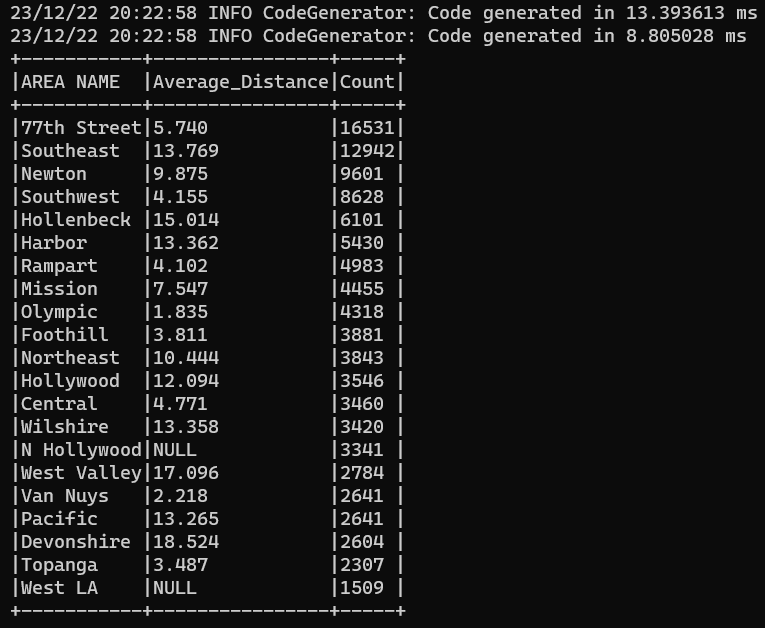
**Ζητούμενο 6.** Για την υλοποίηση του Query 4a δημιουργούμε περίοδο Spark, φορτώνουμε, διαβάζουμε και ενώνουμε τα αρχεία, φιλτράρουμε τα δεδομένα, ώστε να περιλαμβάνονται μόνο περιστατικά που αφορούν πυροβόλα όπλα. Έπειτα, φορτώνουμε τα δεδομένα των αστυνομικών σταθμών και δημιουργούμε ένα λεξικό που να απεικονίζει τις περιοχές των αστυνομικών τμημάτων στις συντεταγμένες τους. Μετά, ορίζουμε τον τύπο Harvesine, ο οποίος υπολογίζει την απόσταση μεγάλου κύκλου μεταξύ δύο σημείων στην επιφάνεια της γης και ορίζουμε μια συνάρτηση που χρησιμοποιεί τον τύπο Harvesine για τον υπολογισμό της απόστασης μεταξύ δύο συντεταγμένων. Τελικά, υπολογίζουμε την απόσταση από την τοποθεσία κάθε εγκλήματος που σχετίζεται με πυροβόλο όπλο έως το πλησιέστερο αστυνομικό τμήμα και εμφανίζουμε τα επιθυμητά αποτελέσματα.

Παρακάτω φαίνεται η υλοποίηση του Query4a χρησιμοποιώντας DataFrame.



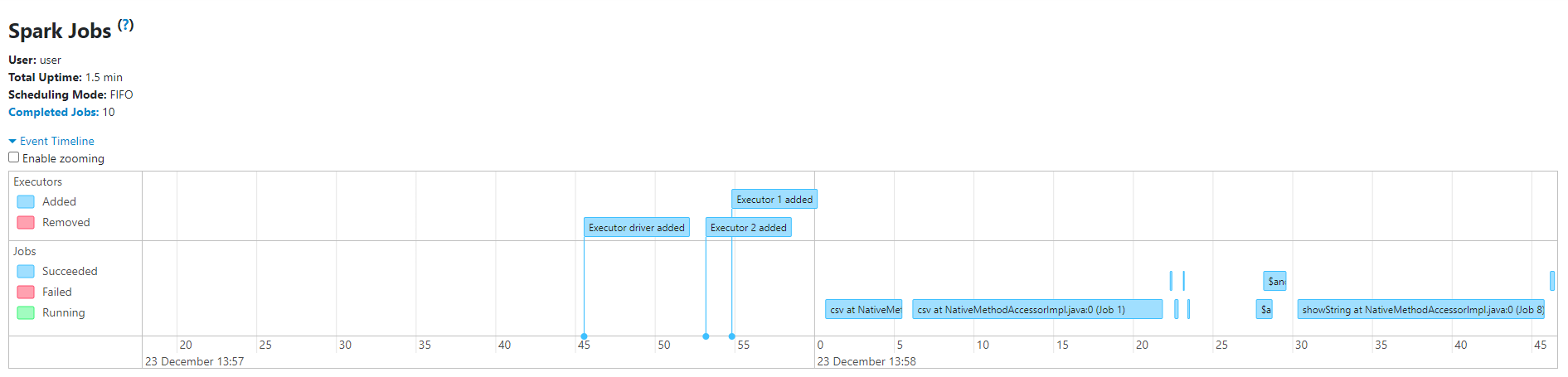
Αντίστοιχα, για την υλοποίηση του Query 4b δημιουργούμε περίοδο Spark, φορτώνουμε, διαβάζουμε και ενώνουμε τα αρχεία, φιλτράρουμε τα δεδομένα, ώστε να περιλαμβάνονται μόνο περιστατικά που αφορούν πυροβόλα όπλα. Η συνάρτηση haversine τροποποιείται για να χειρίζεται τιμές None και αποτελέσματα NaN (Not a Number). Χρησιμοποιούμε τη στήλη AREA NAME από το DataFrame firearm\_crimes και τη στήλη DIVISION από το DataFrame police\_stations για την ένωση. Ειδικότερα, μετατρέπουμε τη στήλη DIVISION σε κεφαλαία γράμματα για να ταιριάζει με τη μορφή του AREA NAME. Τώρα, ο υπολογισμός της απόστασης χρησιμοποιεί απευθείας το γεωγραφικό πλάτος (LAT) και το γεωγραφικό μήκος (LON) από το πλαίσιο δεδομένων firearm\_crimes DataFrame και τις συντεταγμένες (Y, X) από το πλαίσιο δεδομένων police\_stations DataFrame. Τα δεδομένα ομαδοποιούνται με βάση το 'AREA NAME' και τα αποτελέσματα εμφανίζονται με φθίνουσα σειρά.

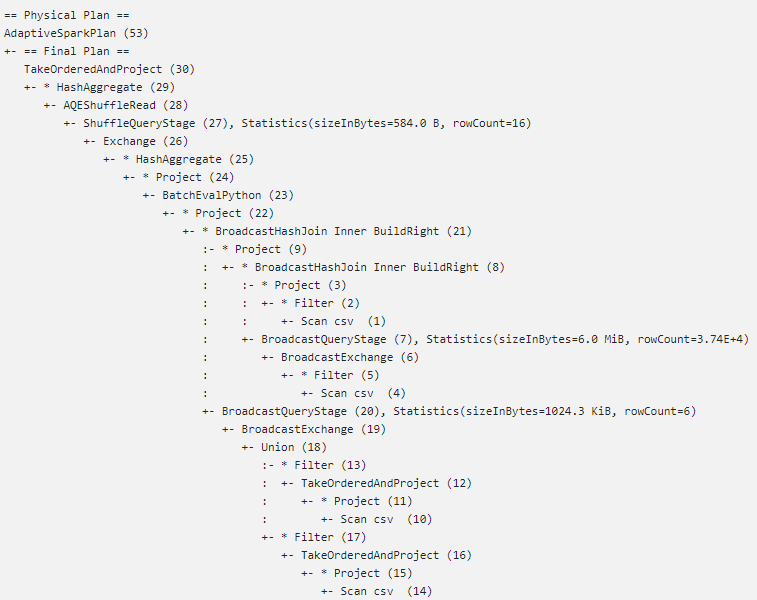
Παρακάτω φαίνεται η έξοδος του Query4b χρησιμοποιώντας DataFrame.

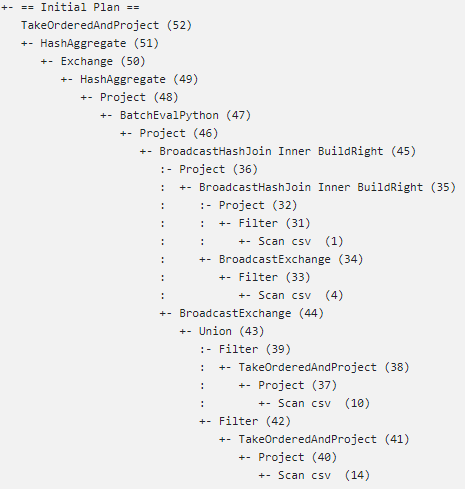


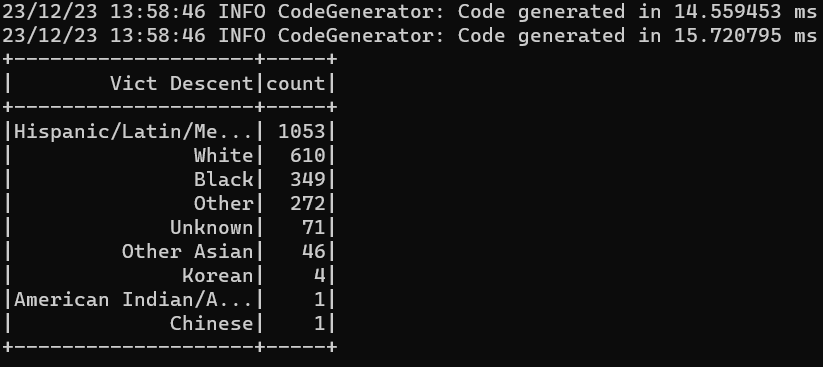
**Ζητούμενο 7.** Στo Query 3 και στο Query 4b έχουμε joins, στα οποία προσθέτουμε την εντολή hint() για το εκάστοτε join, δηλαδή Broadcast, Merge, Shuffle Hash και Shuffle Replicate Nl και την explain() προκειμένου να τυπωθεί ο τρόπος που οργανώνεται η εκτέλεση εσωτερικά του job. Λαμβάνουμε από το Spark UI το γραφικό και περιγραφικό πλάνο της οργάνωσης το οποίο για κάθε περίπτωση παραθέτουμε με εικόνες.

Q3 DataFrame Broadcast 2 Executors

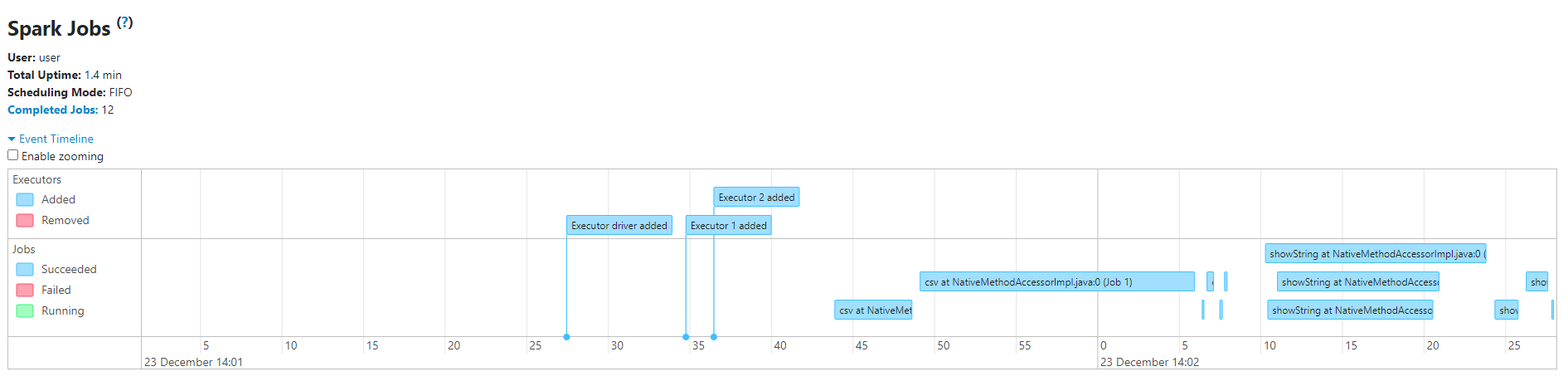




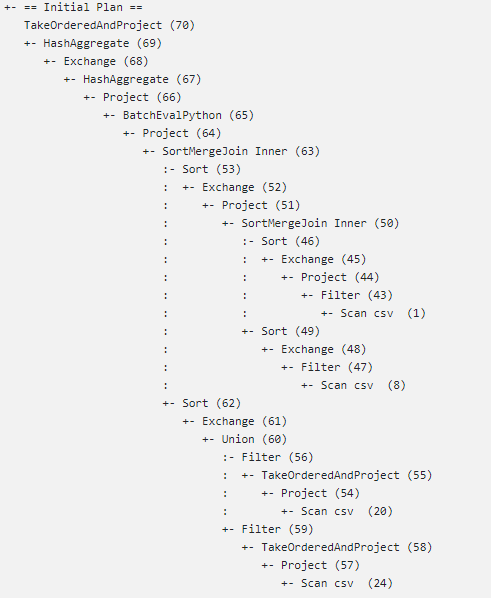


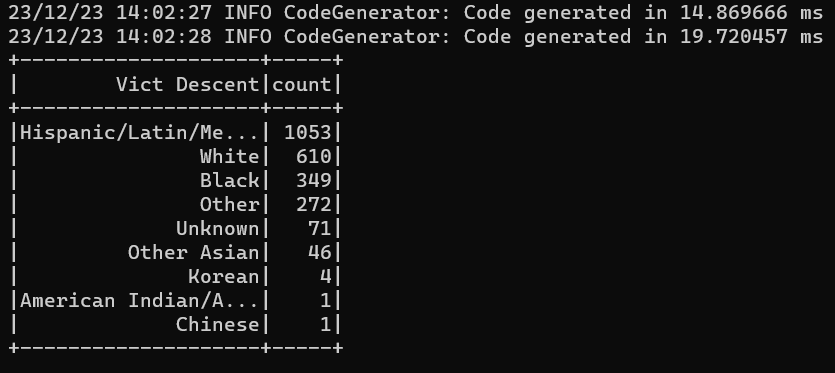


Q3 DataFrame Merge 2 Executors

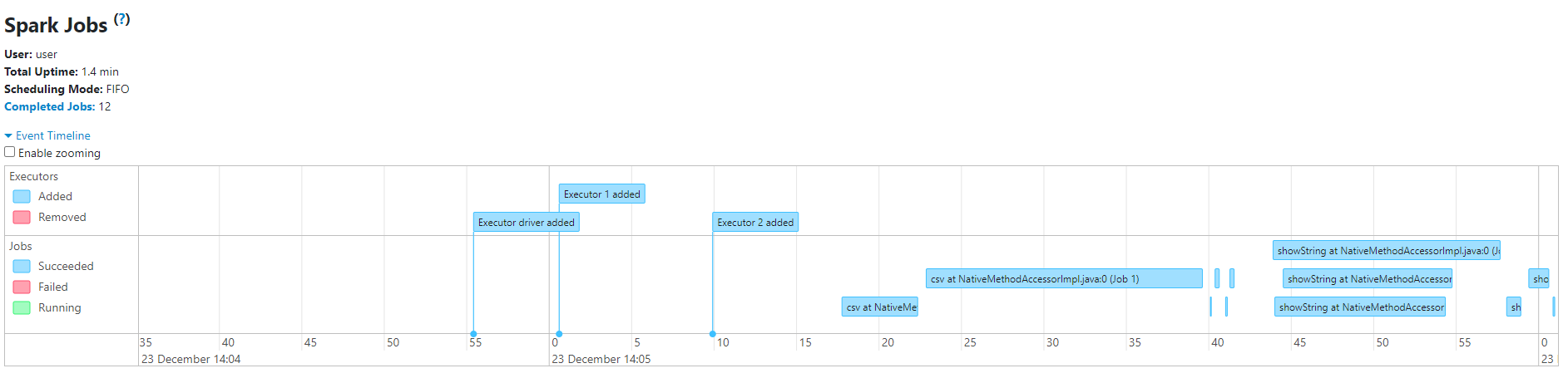




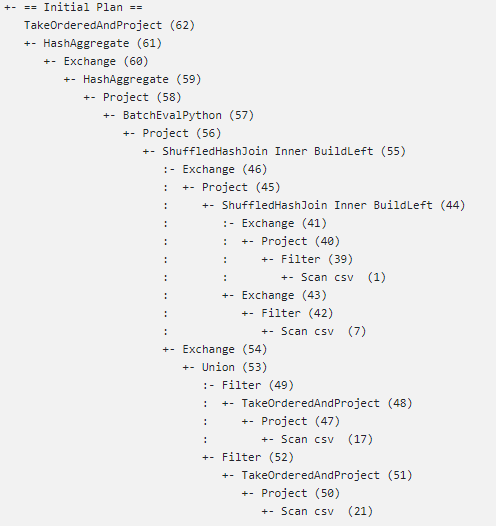


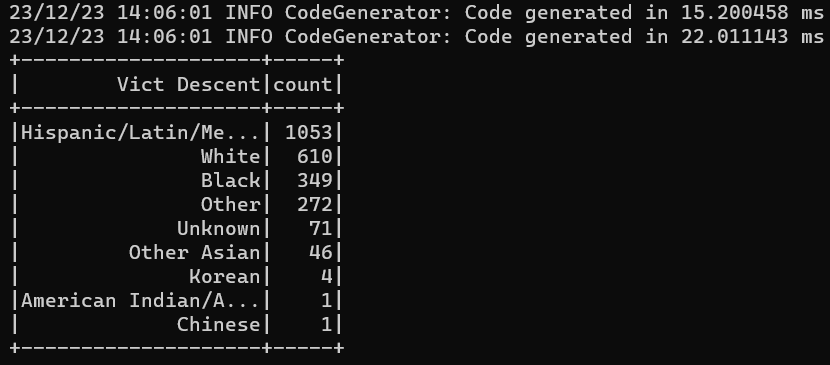


Q3 DataFrame Shuffle Hash 2 Executors

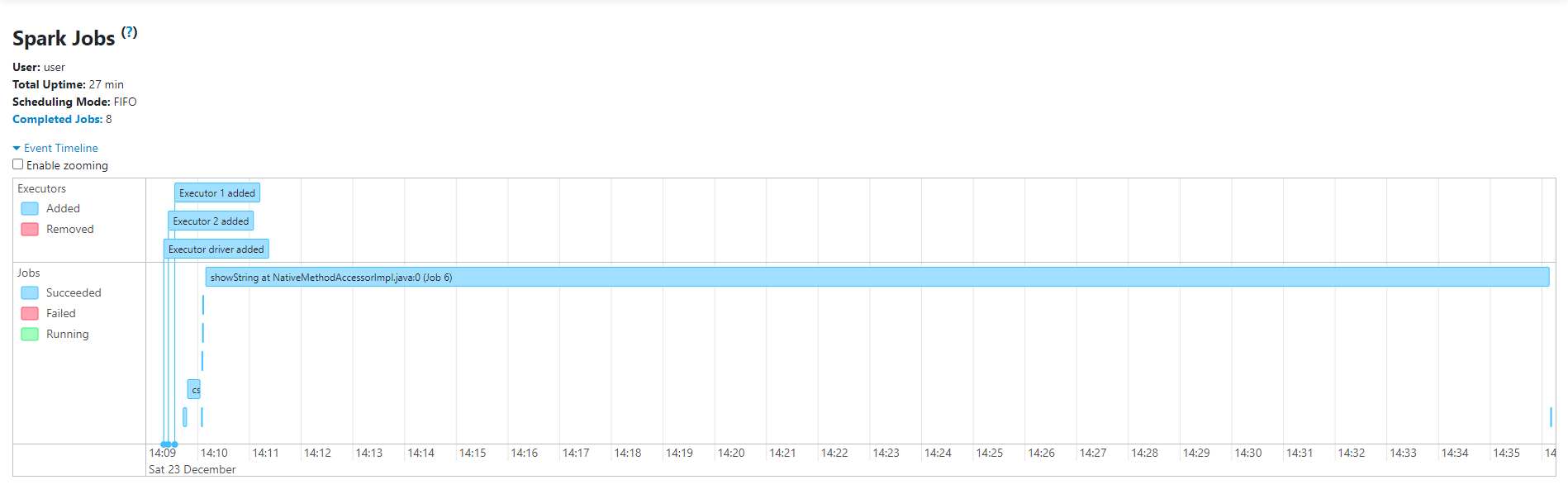


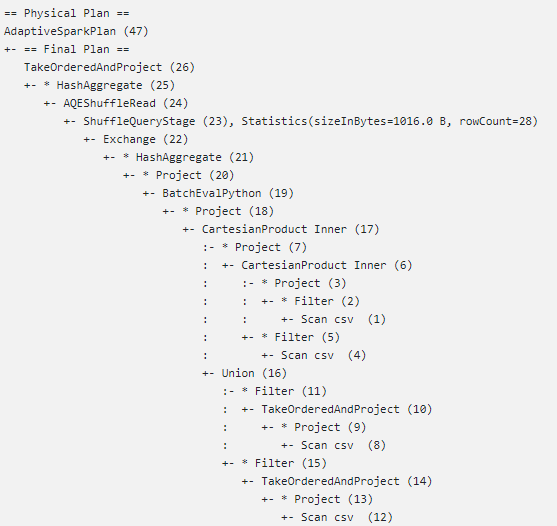


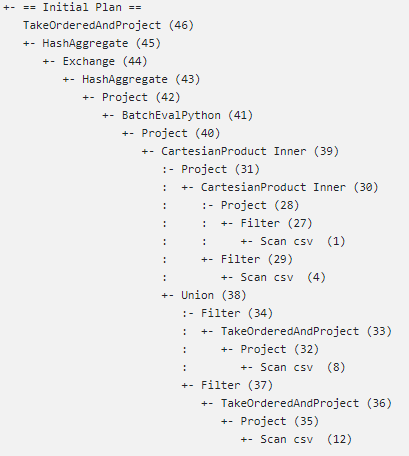


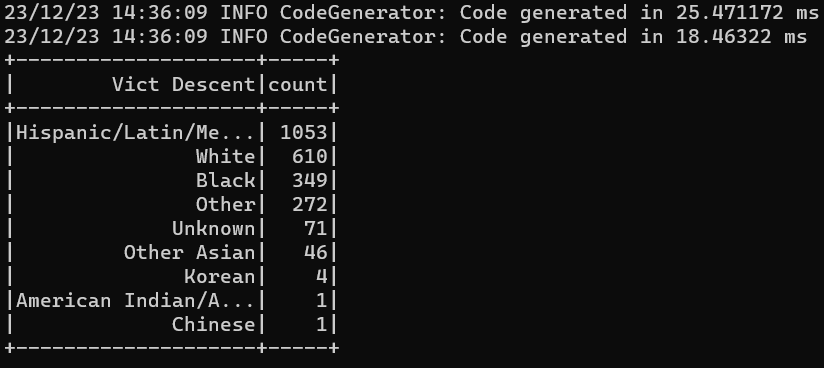


Q3 DataFrame Shuffle Replicate Nl 2 Executors

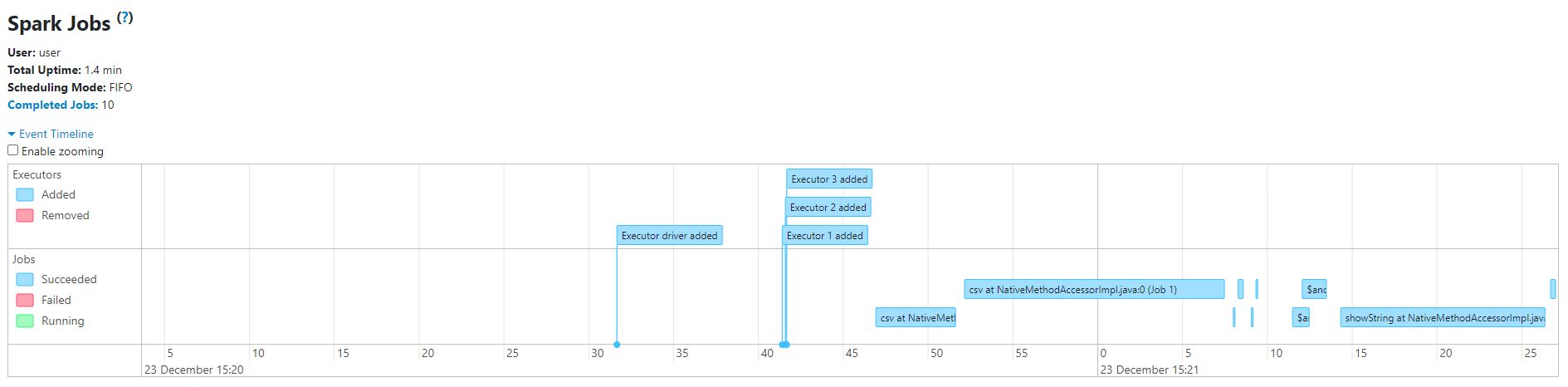


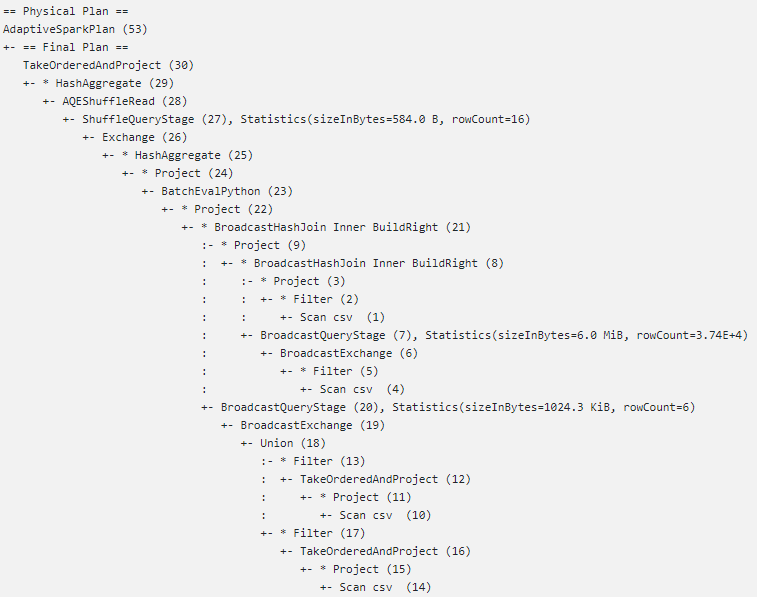


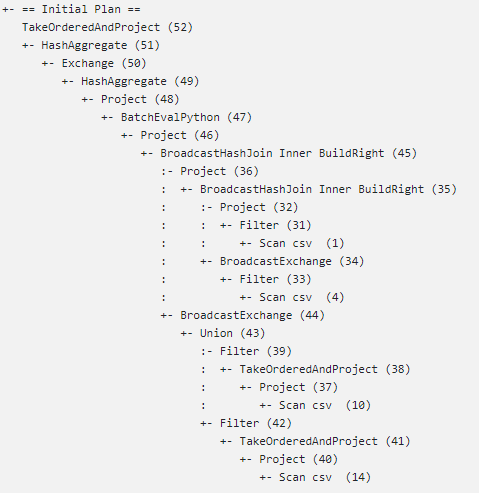




Q3 DataFrame Broadcast 3 Executors

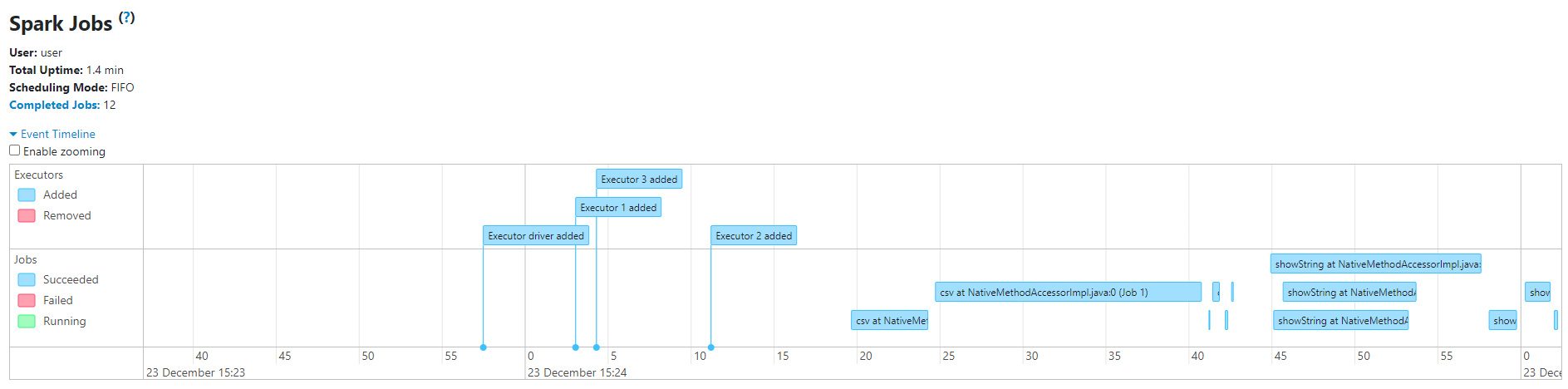




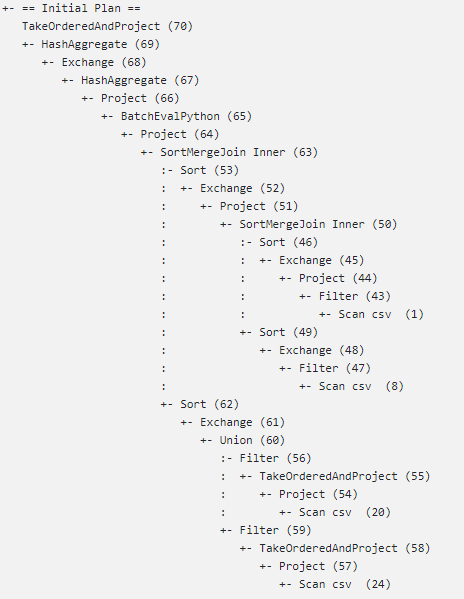




Q3 DataFrame Merge 3 Executors

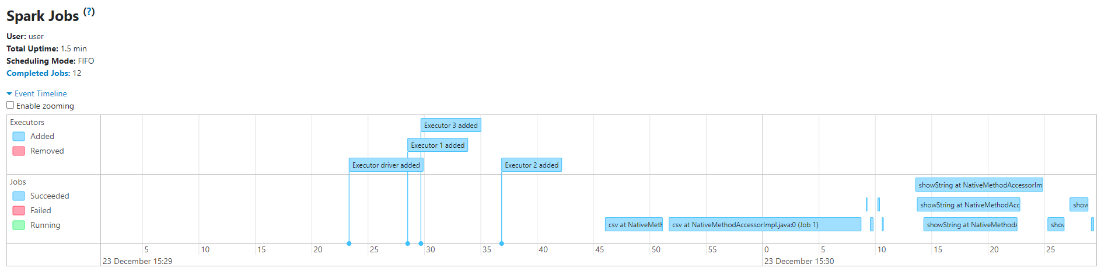




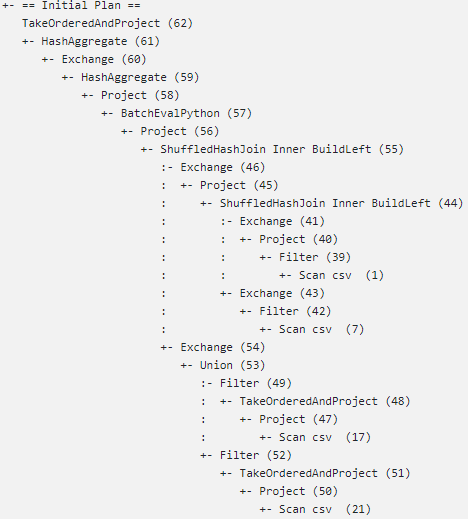




Q3 DataFrame Shuffle Hash 3 Executors

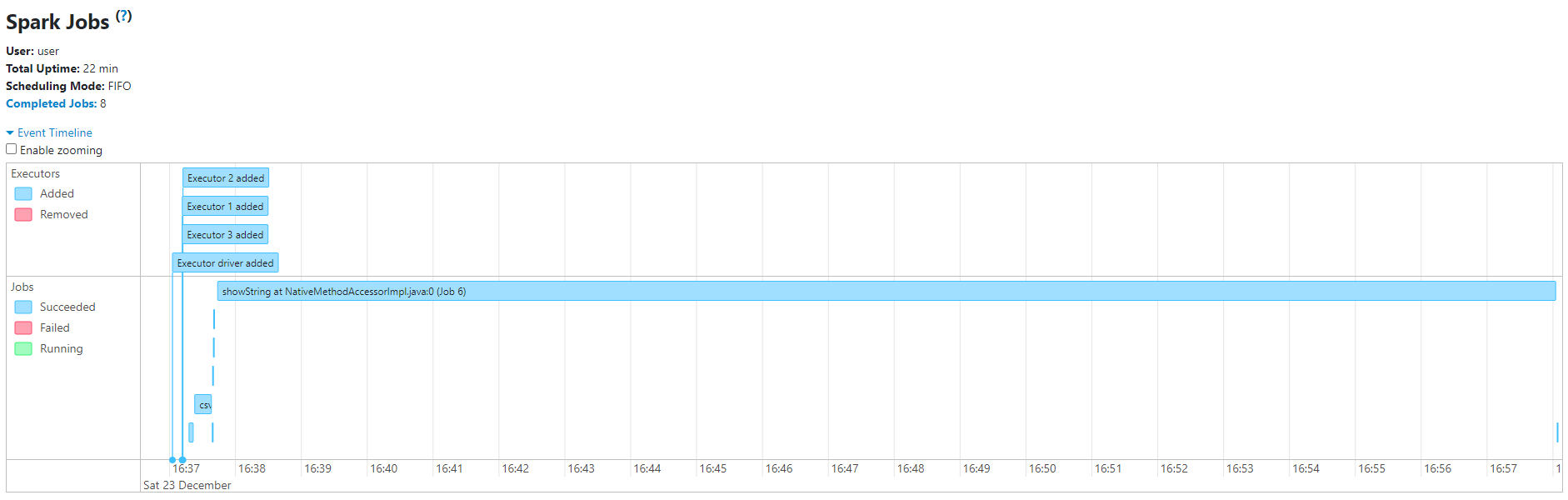


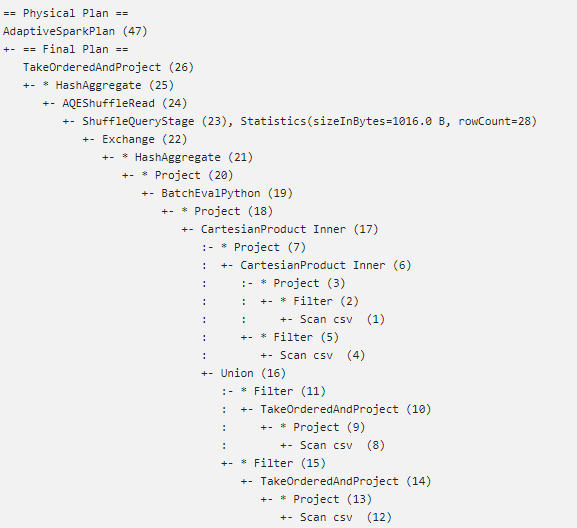


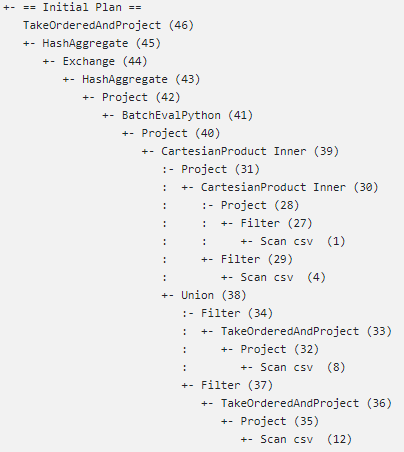




Q3 DataFrame Shuffle Replicate Nl 3 Executors

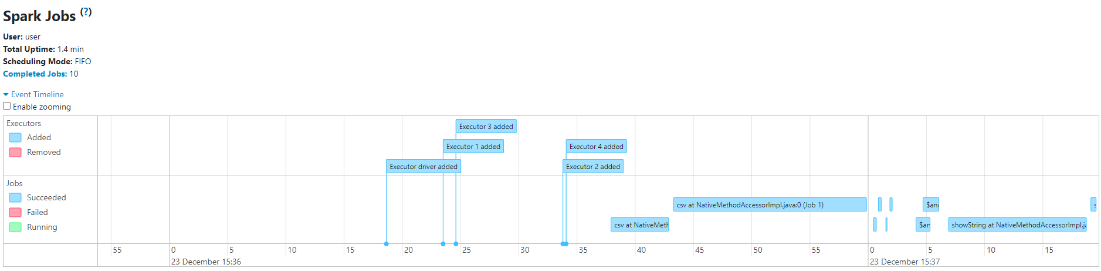


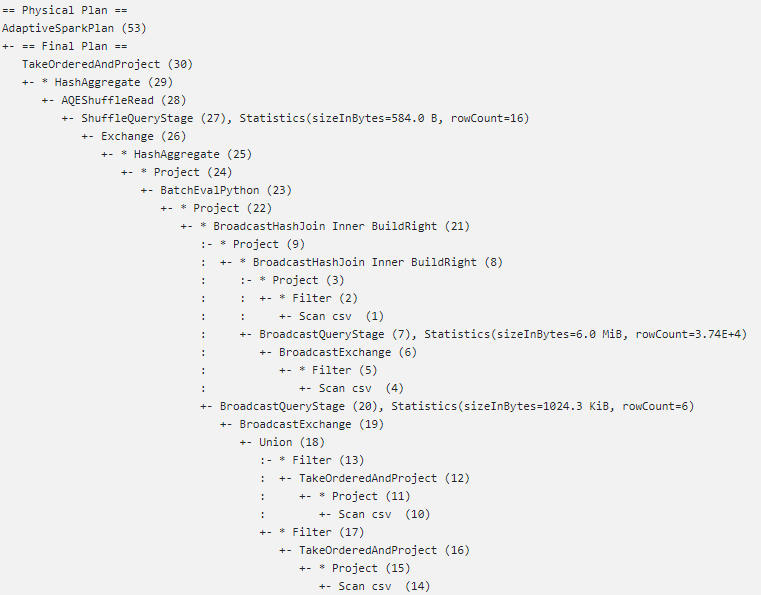






Q3 DataFrame Broadcast 4 Executors

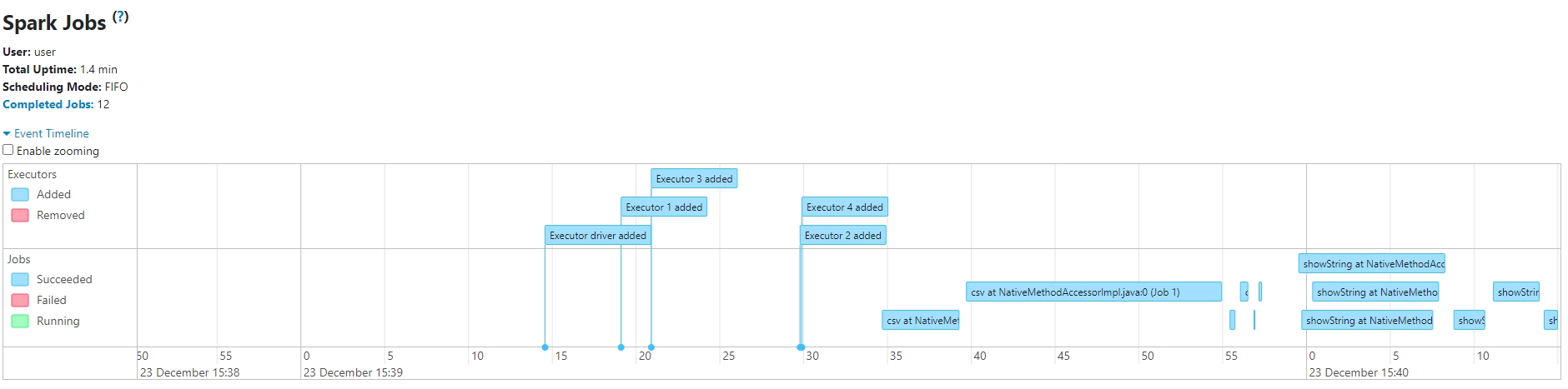




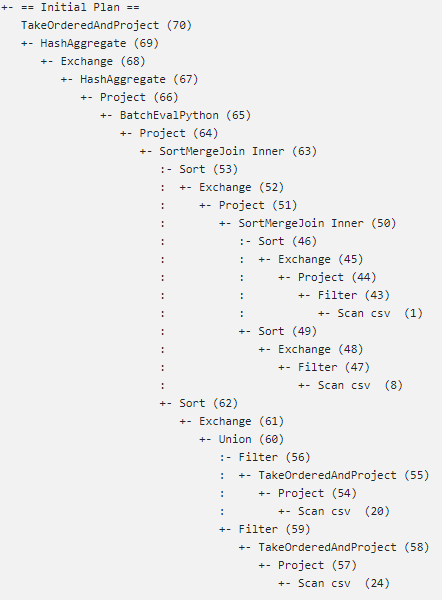




Q3 DataFrame Merge 4 Executors

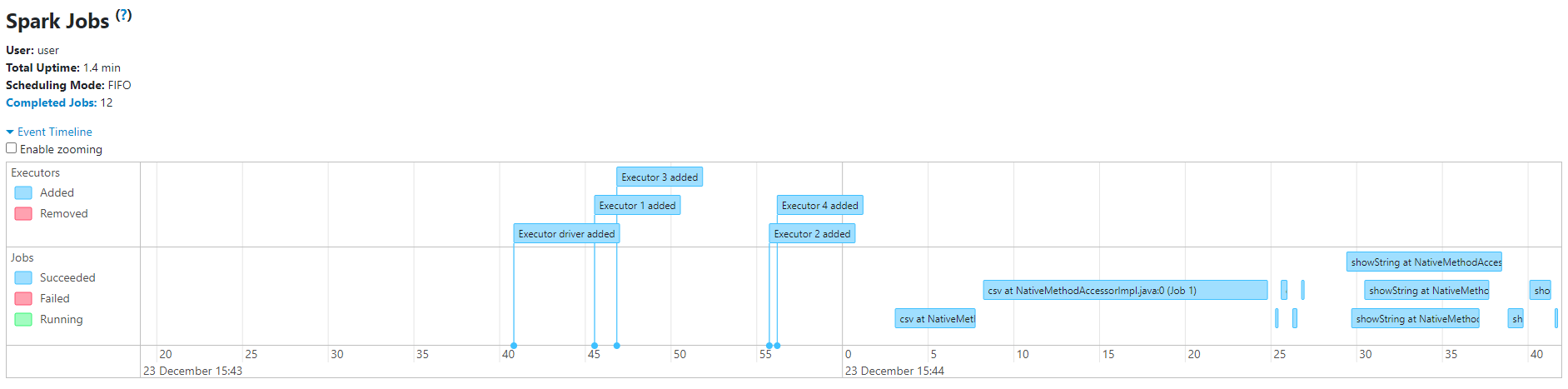




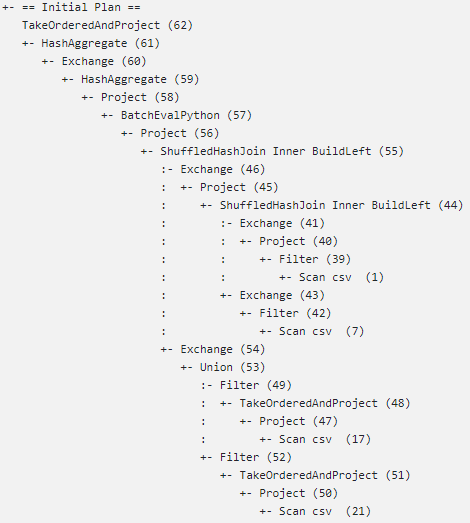




Q3 DataFrame Shuffle Hash 4 Executors

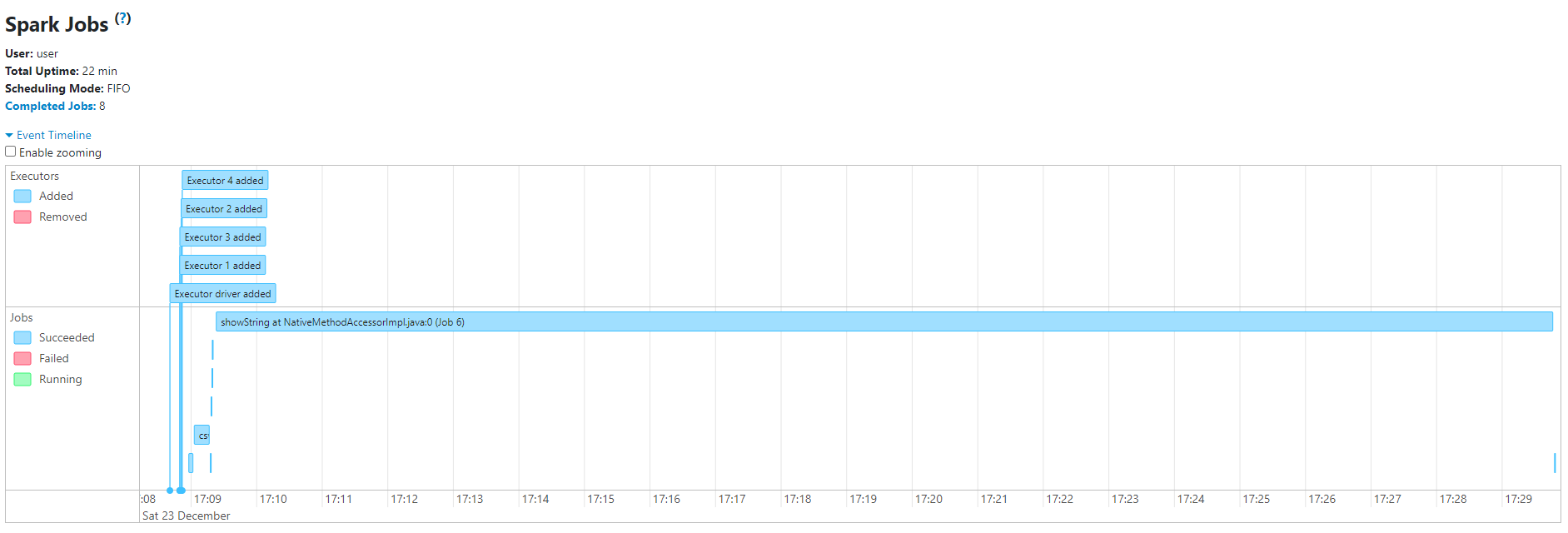


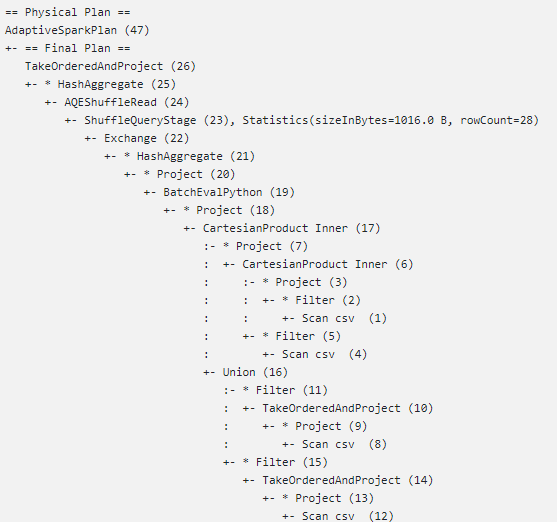






Q3 DataFrame Shuffle Replicate Nl 4 Executors (To Do)

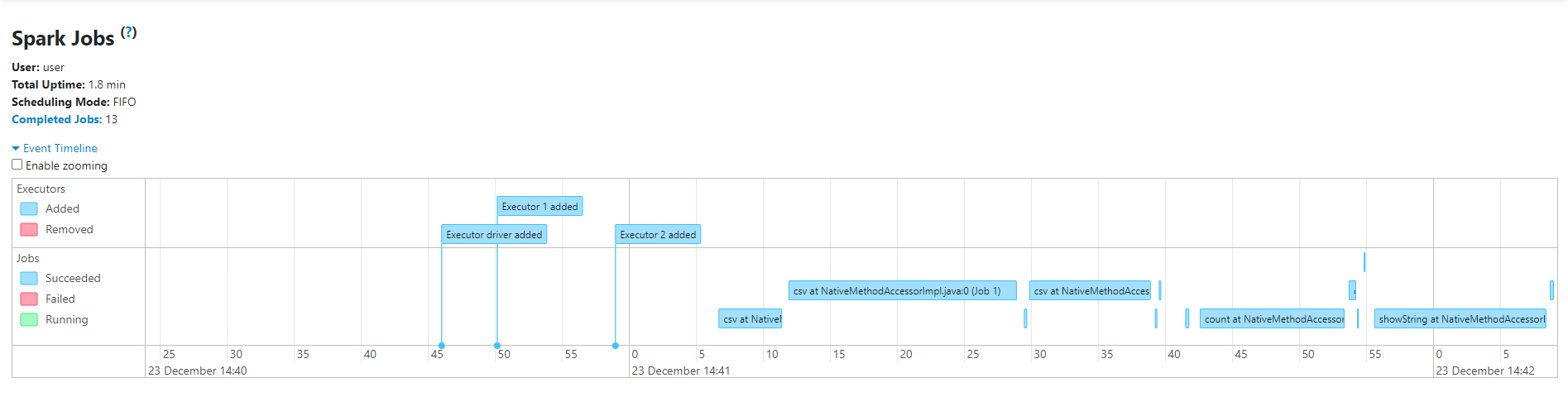


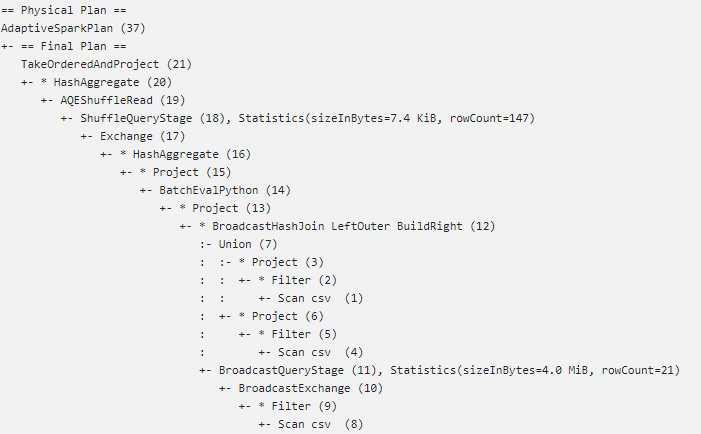


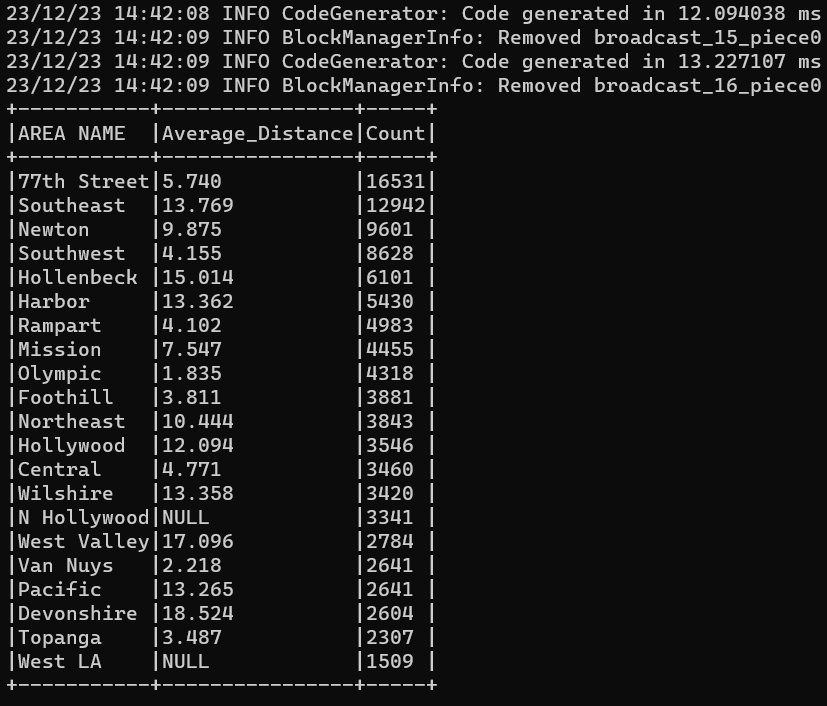
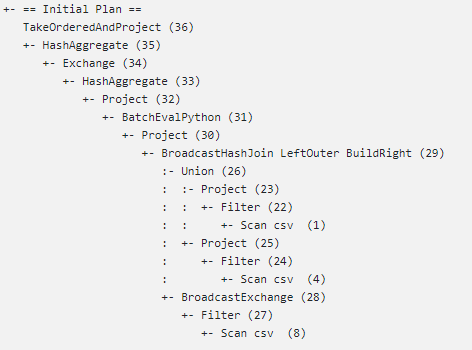




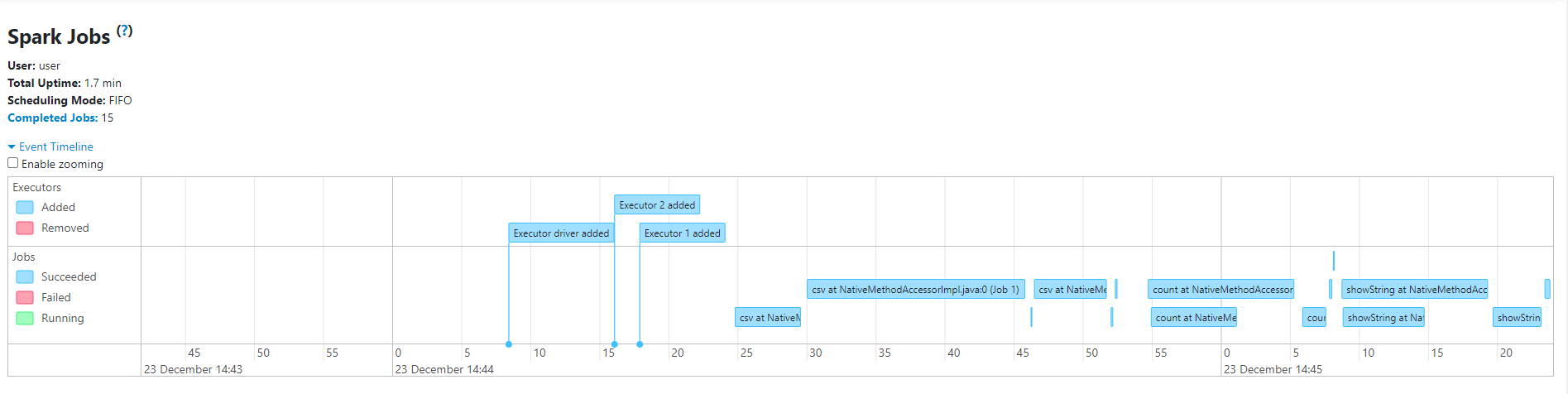
Q4b DataFrame Broadcast

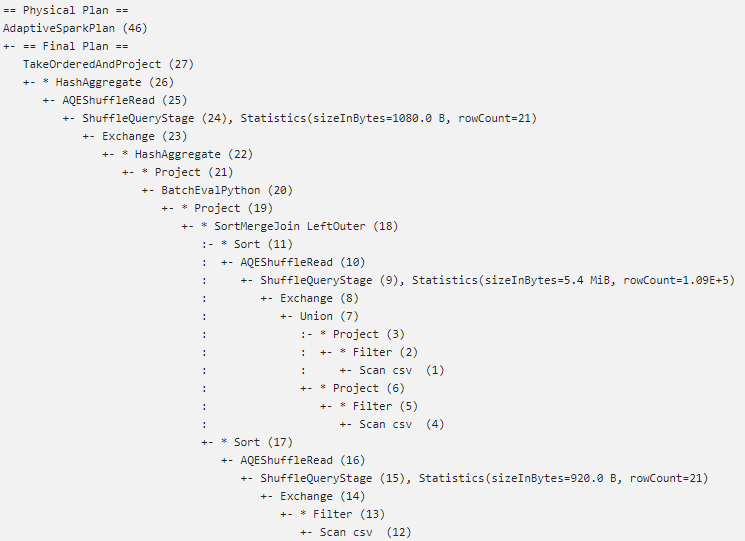




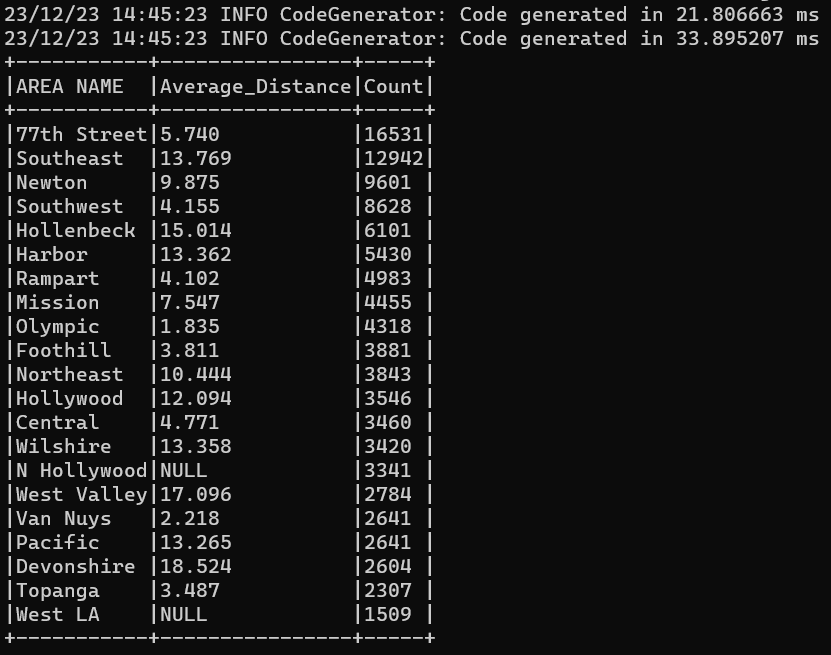


Q4b DataFrame Merge

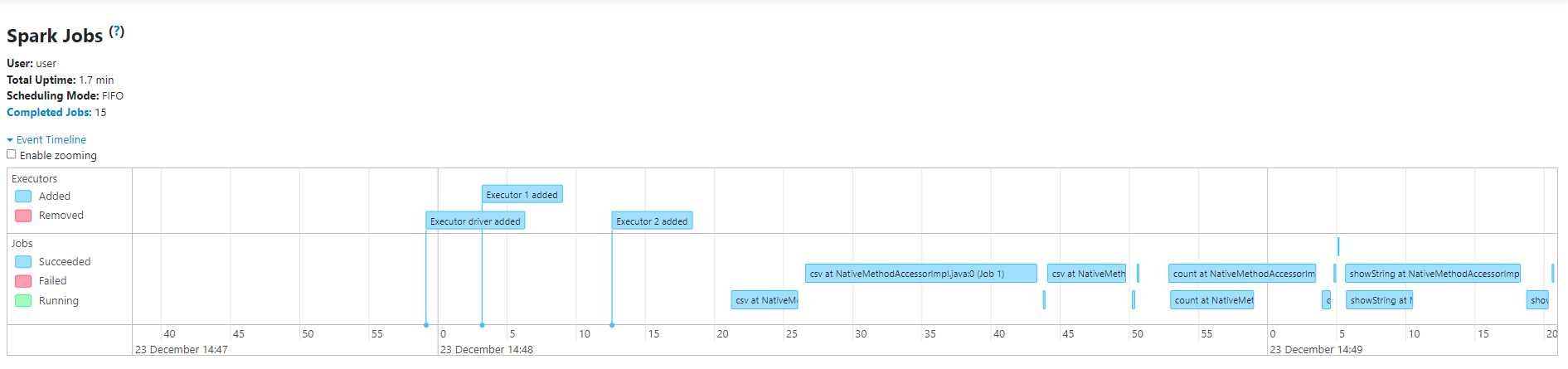


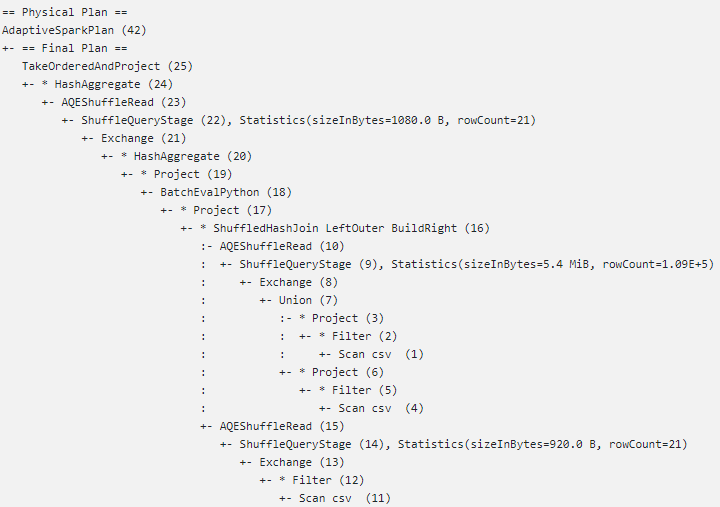






Q4b DataFrame Shuffle Hash

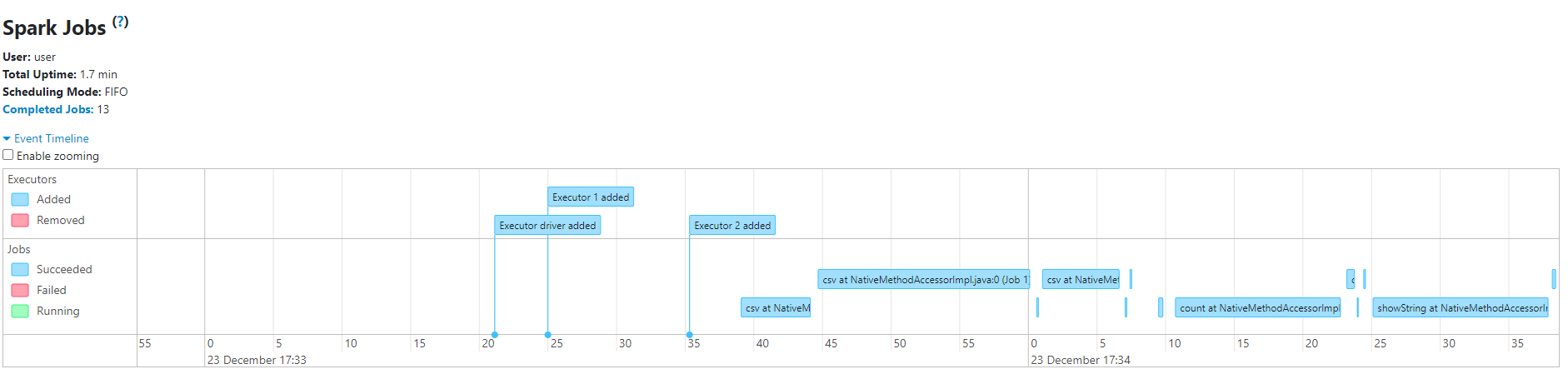


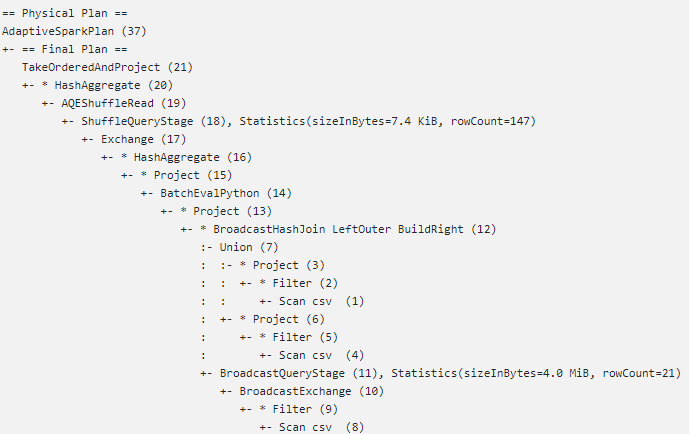


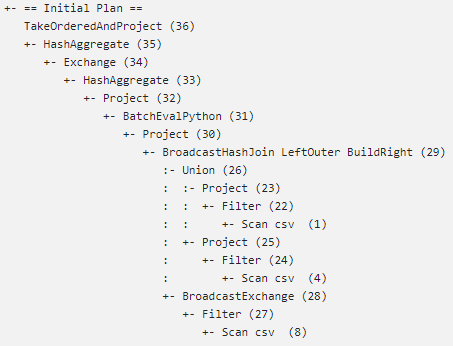




Q4b DataFrame Shuffle Replicate Nl









Broadcast Join:

Αυτή η μέθοδος είναι ιδανική όταν ένα από τα σύνολα δεδομένων είναι πολύ μικρότερο από το άλλο. Το μικρότερο σύνολο δεδομένων μπορεί να χωρέσει στη μνήμη κάθε κόμβου.

Ελαχιστοποιεί την ανακατανομή δεδομένων στο δίκτυο, επειδή το μικρότερο σύνολο δεδομένων μεταδίδεται σε όλους τους κόμβους. Αυτό οδηγεί σε σημαντική βελτίωση των επιδόσεων, ειδικά για μεγάλα σύνολα δεδομένων.

Merge Join:

Η μέθοδος Join είναι γενικά καλή για μεγάλα σύνολα δεδομένων που είναι πολύ μεγάλα για να μεταδοθούν.

Ταξινομεί τα σύνολα δεδομένων με βάση τα κλειδιά σύνδεσης και στη συνέχεια εκτελεί τη συγχώνευση. Αυτή η στρατηγική είναι αποδοτική για μεγάλα σύνολα δεδομένων, αλλά περιλαμβάνει την ανακατανομή των δεδομένων στο δίκτυο, η οποία μπορεί να είναι δαπανηρή.

Shuffle Hash Join:

Χρήσιμη μέθοδος όταν και τα δύο σύνολα δεδομένων είναι μεγάλα αλλά εξακολουθούν να είναι αρκετά μικρά ώστε να χωράνε στη μνήμη όταν κατατμηθούν. Κατακερματίζει τα σύνολα δεδομένων και τα ανακατεύει στους κόμβους.

Shuffle and Replicate Nested Loop Join (Shuffle Replicate NL):

Αυτή η μέθοδος ανακατεύει το ένα σύνολο δεδομένων και αναπαράγει το άλλο για κάθε διαχωριστικό. Είναι γενικά η λιγότερο αποδοτική στρατηγική σύνδεσης και χρησιμοποιείται μόνο για συγκεκριμένες περιπτώσεις όπου άλλες συνδέσεις δεν είναι εφαρμόσιμες.

Στη δική μας περίπτωση, για μεγάλα σύνολα δεδομένων, παρατηρούμε από τα αποτελέσματα που λάβαμε ότι η Broadcast Join και η Merge Join είναι πιο αποδοτικές. Η Shuffle Hash είναι εμφανώς καλύτερη από την Shuffle Replicate NL, εξακολουθούν όμως οι Broadcast και Merge μέθοδοι να είναι οι καλύτερες στην παρούσα εργασία. Αυτό συμβάινει λόγω του όγκου των δεδομένων που έχουμε να επεξεργαστούμε και του τρόπου επεξεργασίας τους κάθε φορά.

Spark DataFrame definition:

DataFrame is a distributed collection of data organized into named columns. Conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs (Resilient Distributed Datasets).

PySpark DataFrame API:

The PySpark DataFrame API, built upon the RDD-based foundation of Spark, offers a Pythonic approach to data manipulation. Its primary goal is to provide a high-level abstraction that is both expressive and easy to use.