Προχωρημένα Θέματα Βάσεων Δεδομένων

Εξαμηνιαία Εργασία

Χρήση του Apache Spark στις Βάσεις Δεδομένων

Ακαδημαϊκό Έτος 2020-2021

Κούστας Κωνσταντίνος - 03115179

Πέππας Αθανάσιος - 03115749

Μέρος 1ο

Ζητούμενο 3

Ερώτημα Q1:

Στο ερώτημα αυτό θα δώσουμε σαν εισοδο στην διαδικασία map μια γραμμή του csv, η οποία στη συνέχεια θα δώσει σαν έξοδο μια τούπλα της οποίας το πρώτο στοιχείο θα είναι η χρονιά ώστε να ομαδοποιηθούν τα στοιχεία ανά χρονιά, και το δευτερο στοιχείο θα είναι μια τούπλα με το όνομα της ταινίας και το κέρδος σύμφωνα με τον ορισμό που μας εχει δωθεί. Τέλος, η διαδικασία reduce θα ομαδοποιήσει τις ταινιες ανα χρονιά και θα εξάγει την ταινία με το μέγιστο κέρδος.

Με βάση αυτά οδηγούμαστε στον παρακάτω ψευδοκωδικα

```
map(line):
    year = line[3]
    name = line[1]
    profit = (line[6]-line[5])/line[5]*100
    emit (year, (name, profit))

reduce(year, list_of_name_profit):
    max = findmax(list_of_name_profit)
    emit (year, max)
```

Σημείωση: Η συνάρτηση findmax παίρνει σαν είσοδο τη λιστα με τις τουπλες, βρίσκει την τουπλα που περιέχει το μέγιστο κέρδος και την επιστρέφει. Δηλαδή, η έξοδος μας είναι της μορφής (year, (name, profit)) ώστε να βολεύει στη συνέχεια να χρησιμοποιήσουμε τη συνάρτηση sortByKey για να ταξινομήσουμε την έξοδο με βάση το έτος.

Ερώτημα Q2:

Στο ερώτημα αυτό θα χρειαστούμε τρεις διαδικασίες map-reduce.

- Η 1η map, θα πάρει σαν ειδοσο ενα line του csv και θα εξαγει ανα χρήστη την βαθμολογία του και εναν ασσο ενώ το reduce θα πάρει ανα χρήστη μια λίστα με βαθμολογίες και τους ασσους και θα υπολογίσει τον συνολικό αριθμό των βαθμολογιών καθώς και το πλήθος τους.
- Η 2η map-reduce χρησιμεύει στην ομαδοποίηση των δεδομένων. Συγκεκριμένα, για κάθε χρήστη κοιτάει τη μέση βαθμολογία και αν ειναι κατω από 3 του δείνει id = 1 ενω σε αντίθετη περίπτωση, id=2. Έτσι, καταλήγουμε σε ενα dataset με δύο τουπλες: (1, πληθος1) και (2, πληθος2).
- Τέλος, η 3η και τελευταία map-reduce χρησιμεύει ώστε να έχουμε κατω από το ίδιο id τις δύο μεταβλητές ώστε να υπολογίσουμε το τελικό ποσοστό.

Παρακάτω φαίνεται ο ψευδοκώδικας

```
map1(line):
    user = line[0]
    rating = line[2]
    emit(user, (rating,1))

reduce1(user, list_of_ratings_aces):
    r = 0
    c = 0
    for x, y in list_of_ratings_aces:
        r += x
        c += y
    average = r/c
    emit(user,average)

map2(user,average):
    if average <= 3: return (1,1)
    else return (2,1)

reduce2(id,list_of_aces):
    c = len(list_of_aces)
    emit(id,c)

map3(id,c):
    emit(1,c)
```

```
reduce3(1,list_of_c):
    c1 = list_of_c[0]
    c2 = list_of_c[1]
    per = c2/(c1+c2)*100
    emit(per)
```

Σημείωση: Στην reduce3 υποθέτουμε πως στην πρώτη θέση είναι το πληθος όσων έχουν μέση βαθμολογία κάτω από 3.

Ερώτημα Q3:

Σε αυτή την περίπτωση ακολουθούμε τον τύπο που μας εχει δωθεί ως εξής:

- Στο 1ο map-reduce επενεργούμε στον πίνακα ratings και βρίσκουμε τη μέση βαθμολογία για κάθε ταινία
- Στο 2ο map επενεργούμε πάνω στον πίνακα movie_genres για να έχουμε τα δεδομένα σε καταλληλη μορφή ώστε στη συνέχεια να κάνουμε join με τον πίνακα ratings. Η μορφή και των δύο πινάκων θα πρέπει να είναι (id_movie, records) ώστε να μπορέσουμε να τους συνενόσουμε με βάση την ταινία
  Κάνουμε το join και έχουμε ένα αρχείο data που περιέχει σε κάθε γραμμή την εξής τούπλα:
  (id_movie, (list_of_genres,average))
- Η 3η και τελευταία map-reduce για κάθε είδος ταινίας υπολογίζει τη μέση βαθμολογία.

```
map1(line):
    id_movie = line[1]
    rating = line[2]

    emit(id_movie, (rating, 1))

reduce(id_movie, list_of_ratings):
    sr = 0
    sc = 0

    for x,y in list_of_ratings:
        sr += x
        sc += y
    av = x/y

    emit(id_movie, av)

map2(line):
    id_movie = line[0]
    genre = line[1]

    emit(id_movie, genre)

join(ratings,movie_genres)

map3(line):
    average = line[1][1]
    list_of_genres = line[1][0]
    for g in list_of_genres:
        emit(g,(average,1))

reduce3(genre, list_of_avg):
```

```
    savg = 0
    scount = 0
    for x,y in list_of_avg:
        savg += x
        scount += y
    totalavg = x/y
    emit(g, totalavg)
```

Ερώτημα Q4:

Στο ερώτημα αυτό αρχίκα χρησιμοποιούμε 2 διαδικασίες map-reduce.

- Στην 1η κάνουμε ένα map στα αρχεία movies και genre_movies εν τέλει τα δεδομένα είναi (id_movie, length, year) , (id_movie) αντίστοιχα. Ενώ στη συνέχεια μέσω του reduce γίνεται το επιθυμητό join και τα δεδομένα μας είναι (id_movie, (length,year)). Εδώ να σημειώσουμε πως το join γίνεται μόνο σε κοινά κλειδία συνεπώς δε χρειάζεται να κανουμε ιδιαίτερη μέριμνα για τα κλειδιά που περισσεύουν από το αρχείο movies
- Τέλος με τη 2η διαδικασία map-reduce βρίσκουμε ανα 5ετία το μέσο μήκος περιληψης. Πιο συγκεκριμένα:
    - Με τη map ταξινομούμε κάθε ταινία σε μια 5ετία και κάνουμε emit την 5ετια το μέγεθος της περίληψης και εναν άσσο
    - Με τη reduce για κάθε 5ετία αθροίζουμε τα μήκη και τους άσσους βρισκουμε το μέσο όρο και εξάγουμε το τελικό αποτέλεσμα

Έτσι έχουμε:

```
map1(line: record from either movies or movie_genres):
    if line.belongs(movies.csv):
        id_movie = line[0]
        length = len(line[2])
        year = line[3]
        emit(id_movie, (length,year))
    else:
        if line[1]=='Drama':
            id_movie = line[0]
            emit(id_movie)

reduce1(id, list_of_values):                            // list_of_values:
(id_movie, (length,year))
    emit(id, list_of_values)

map2(line: record from joined dataset):
    length = line[1][0]
    year = line[1][1]

    if year.belongs(2000,2004):
        emit('2000-2004',(length,1))
    elif year.belongs(2005,2009):
        emit('2005-2009',(length,1))
    elif year.belongs(2010,2014):
        emit('2010-2014',(length,1))
    else:
        emit('2015-2019',(length,1))
```

```
reduce2(id, list_of_lengths):
    s_length = 0
    scount = 0
    for x,y in list_of_lengths:
        s_length += x
        scount += y
    avg = s_length/scount
    emit(id, avg)
```

Ερώτημα Q5:

Στο ερώτημα αυτό θα χρειαστούμε τα δεδομένα και από τα τρια αρχεία. Αυτό σημαίνει πως θα κάνουμε join τα τρια datasets. Πιο συγκεκριμένα, θα κάνουμε join με κλειδί συνένωσης το id_movie. Έτσι, αρχικά το κάθε dataset γίνεται:

- *movies : (id_movie, (title, popularity))*
- *movie_genres: (id_movie, genre)*
- *ratings: (id_movie, (user, rating))*

Και στη στυνέχεια, μετά από το join, έχουμε το εξης dataset: (*id_movie*, (((*title, popularity*) , *genre*), (*user, rating*))

Στη συνέχεια για να απαντήσουμε στο query χρησιμοποιούμε 3 διαδικασίες map-reduce. Πιο συγκεκριμένα:

- Το 1ο ομαδοποιεί τα δεδομένα με βάση την τούπλα (genre,user). Έτσι για κάθε δυάδα είδους, χρήστη έχουμε μια λίστα με τίτλο βαθμολογία και δημοτικότητα
- Το 2ο ομαδοποιεί τα δεδομένα με βάση το είδος, υπολογίζει το πλήθος των βαθμολογιών που εχει δώσει κάθε χρήστης και κρατάει αυτόν με τις περισσοτερες
- Στο 3ο βρίσκουμε την ταινία με τη μέγιστη και την ελάχιστη βαθμολογία και τα ταξινομούμε με βάση το όνομα του είδους.

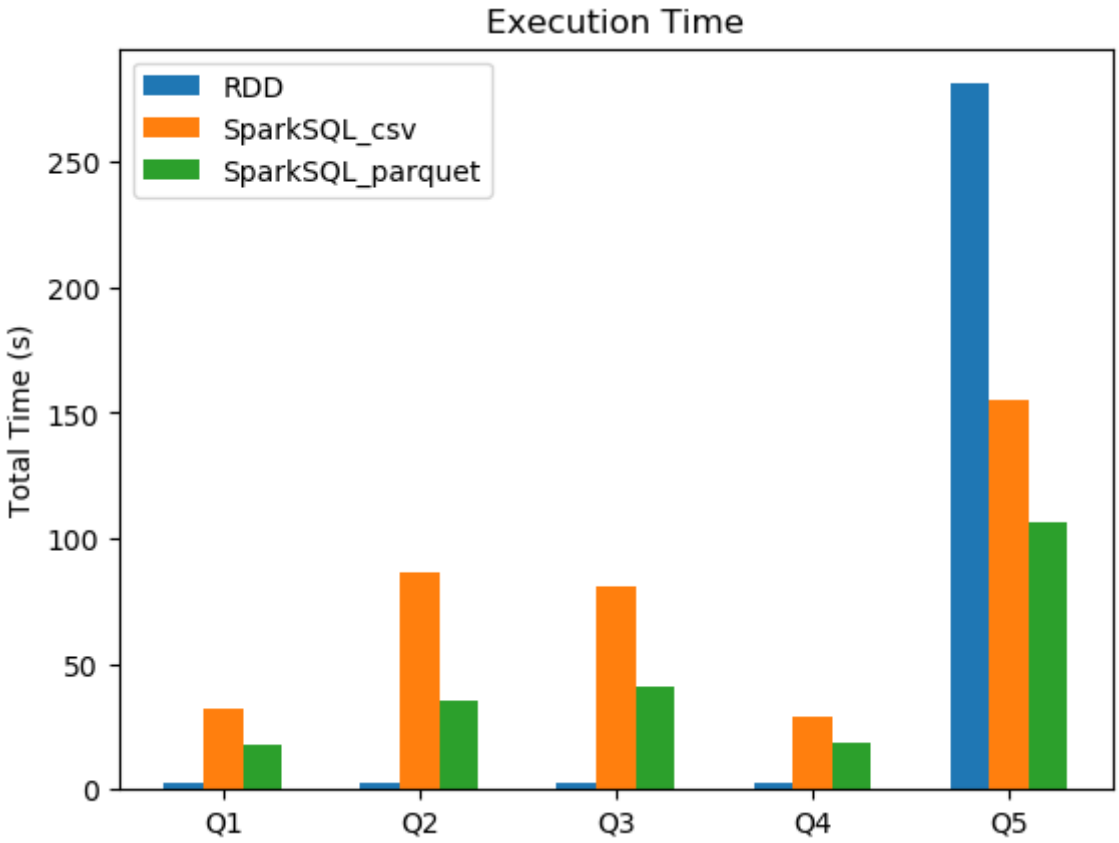
```
map1(line:record from movies, movie_genres or ratings):
    if line.belongs(movies.csv):
        id_movie = line[0]
        title = line[1]
        popularity = line[7]
        emit(id_movie, (title, popularity))
    elif line.belongs(movie_genres.csv):
        id_movie = line[0]
        genre = line[1]
        emit(id_movie, genre)
    else:
        id_movie = line[1]
        user = line[0]
        ratings = line[2]
        emit(id_movie,(user,rating))

reduce1(id_movie, list_of_values):
    emit(id_movie, list_of_values)

map2(line: record from joined dataset)              //dataset: ((id_movie, (
((title, popularity) , genre), (user, rating)):
    genre = line[1][0][1]
    user = line[1][1][0]
    title = line[1][0][0][0]
```

```
    rating = line[1][1][1]
    popularity = line[1][0][0][1]

    emit ((genre,user), (title,rating,popularity))

reduce2(key, list_of_values):
    emit(key, list_of_values)

map3(key, values):
    genre = key[0]
    user = key[1]
    rat_count = len(values)/3                        // Εδω ουσιαστικά ειναι μια
λιστα που ανα 3 στοιχεία αναφερόμαστε στην ίδια
// βαθμολογία αρα αμα διαιρέσουμε το μήκος με το 3 θα βρούμε το
// ζητούμενο πλήθος
    emit (genre, (user, values, rat_count))

reduce3(genre, list_of_values):
    max = 0
    muser = list_of_values[0]
    mvalues = list_of_values[1]
    for u, v, c in list_of_values:
        if c >= max:
            max = c
            muser = u
            mvalues =v
    emit (genre, (muser,mvalues))

map4(genre, (user,values)):                          // valeus: (title, rating,
popularity)
    dct = {values[i+1]: (values[i],values[i+2]) for i in range(0, len(values),
3)}
    M = (dct[max(dct)][0],dct[max(dct)][1],max(dct))
    m = (dct[min(dct)][0],dct[min(dct)][1],min(dct))
    dM = {}
    dm = {}
    for i in range(0, len(values), 3):
        if M[2]==values[i+1]:
            dM[values[i+2]] = (values[i],values[i+1])

        if m[2]==values[i+1]:
            dm[values[i+2]] = (values[i],values[i+1])

    M = dM[max(dM)]
    m = dm[max(dm)]

reduce4(genre, values):
    data = (genre, values)
    data.sortByKey()
    emit(data)
```

Ζητούμενο 4

Μετά από την εκτέλεση των ερωτημάτων έχουμε τους εξής χρόνους εκτέλεσης σε seconds:

Query	RDD	SparkSQL CSV	SparkSQL PARQUET
Q1	2.610966682434082	32.420072078704834	17.7742121219635
Q2	2.666517496109009	86.39740824699402	35.41659474372864
Q3	2.5541470050811768	80.57300066947937	41.31655311584473
Q4	2.4377427101135254	29.07029914855957	18.80320143699646
Q5	281.087694644928	154.98738551139832	106.39897274971008



Σχόλια:

- Στα 4 πρώτα ερωτήματα η πιο γρήγορη υλοποίηση είναι αυτή του map-reduce. Ενα αναμενόμενο αποτέλεσμα καθώς η τεχνική αυτή είναι προσαρμοσμένη στην επίλυση ερωτημάτων σε μεγάλο όγκο δεδομένων.
- Στα ερωτημάτα 2 και 4 βλέπουμε πως έχουμε μια αύξηση στους χρόνους εκτέλεσης. Το γεγονός αυτό οφείλεται στο αρχείο ratings που περίεχει πολύ περισσότερες εγγραφες από τα άλλα δύο αρχεία.
- Στο ερώτημα 5, εχουμε πολύ μεγαλύτερους χρόνους εκτέλεσης. Αυτό οφείλετε στο γεγονός πως επεξεργαζόμαστε και τα 3 αρχεία κατα συνέπεια και το ratings που είδαμε πιο πάνω πως δυσχεραίνει την κατάσταση και επίσης έχουμε 2 τουλάχιστον συνενώσεις.
- Σε όλα τα ερωτήματα η χρήση του αρχείου parquet έναντι του csv, μειώνει το χρόνο εκτέλεσης. Αυτό συμβαίνει διότι στο csv χρησιμοποιούμε το infer schema που ναι μεν διευκολύνει την

ανάγνωση των δεδομένων αλλά ταυτόχρονα την καθυστερεί διότι το διαβάζει μια επιπλέον φορά. Αυτό στο parquet δεν χρειάζεται συνεπώς ο χρόνος εκτέλεσης είναι μικρότερος.

Για να κατανοήσουμε για ποιο λόγο δε χρησιμοποιείται από τα αρχεία parquet το infer schema πρώτα πρέπει να δούμε τι ειναι και πώς διαφέρουν τα csv από τα parquet

Το infer schema αναφέρεται στους τυπους δεδομένων ώστε να γίνει καλύτερη ανάγνωση και να παράξει σωστά αποτελέσματα. Η διαφορά των αρχείων parquet σε σχέση με τα csv έγγυται στο γεγονός οτι τα πρώτα κρατάνε έξτρα πληροφορία για τα ίδια τα δεδομένα ώστε να διευκολύνεται και να είναι πιο γρήγορη η ανάγνωσή τους.

Συνεπώς, όταν διαβάζουμε ένα αρχείο parquet, δεν εχουμε ανάγκη από infer schema καθώς το ίδιο το αρχείο περιέχει ολη την πληροφορία που χρειαζόμαστε.

Γενικά σχόλια

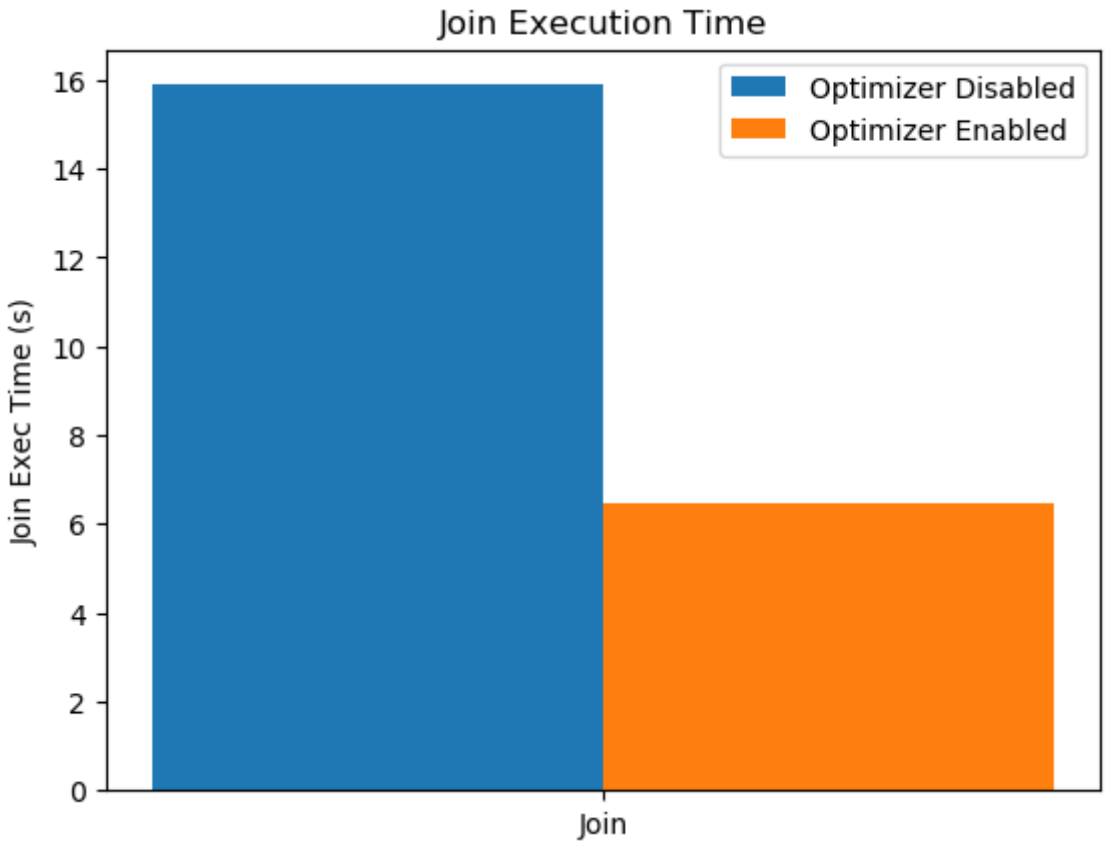
- Στο q1 σε περίπτωση ισοβαθμίας εμφανίζουμε αυθαίρετα το ενα αποτέλεσμα.
- Στο q3 δε λαμβάνουμε υπόψη εγγραφές χωρίς βαθμολογία και επίσης κάνουμε ειδική μέριμνα για διπλότυπες εγγραφές στο αρχείο movie_genres
- Στο q4 δε λαμβάνουμε υπόψη ταινίες με κενές περιλήψεις και το φιλτράρισμα γίνεται ως εξής: με βάση τα κενά λαμβάνουμε όμως και τα διπλά καθώς επίσης και τα σημεία στίξης.
- Στο q5 αν έχουμε ισοβαθμία μεταξύ χρηστών εμφανίζουμε αυθαίρετα τον έναν.

Μέρος 2ο

Ζητούμενο 4

Η συμπλήρωση στον κώδικα που κάναμε ώστε να απενεργοποιηθεί ο βελτιστοποιητής είναι ότι προσθέσαμε το configuration `spark.sql.autoBroadcastJoinThreshold` με τιμή `-1` ώστε όποιο και να είναι το μέγεθος του αρχείου να μη γίνει broadcast.

Οι χρόνοι εκτέλεσης που πήραμε φαίνονται στο παρακάτω ραβδόγραμμα



Ενώ το πλάνο εκτέλεσης που παράγει ο βελτιστοποιητής σε κάθε περίπτωση φαίνεται στον παρακάτω πίνακα

Disabled	Enabled
*(6) SortMergeJoin [id_movie#8L], [id_movie#1L], Inner :- *(3) Sort [id_movie#8L ASC NULLS FIRST], false, 0 : +- Exchange hashpartitioning(id_movie#8L, 200) : +- *(2) Filter isnotnull(id_movie#8L) : +- *(2) GlobalLimit 100 : +- Exchange SinglePartition : +- *(1) LocalLimit 100 : +- *(1) FileScan parquet [id_movie#8L,genre#9] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/movie_genres.parquet], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<id_movie:bigint,genre:string> +- *(5) Sort [id_movie#1L ASC NULLS FIRST], false, 0 +- Exchange hashpartitioning(id_movie#1L, 200) +- *(4) Project [user#0L, id_movie#1L, rating#2, timestamp#3L] +- *(4) Filter isnotnull(id_movie#1L) +- *(4) FileScan parquet [user#0L,id_movie#1L,rating#2,timestamp#3L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/ratings.parquet], PartitionFilters: [], PushedFilters: [IsNotNull(id_movie)], ReadSchema: struct<user:bigint,id_movie:bigint,rating:double,timestamp:bigint>	*(3) BroadcastHashJoin [id_movie#8L], [id_movie#1L], Inner, BuildLeft :- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])) : +- *(2) Filter isnotnull(id_movie#8L) : +- *(2) GlobalLimit 100 : +- Exchange SinglePartition : +- *(1) LocalLimit 100 : +- *(1) FileScan parquet [id_movie#8L,genre#9] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/movie_genres.parquet], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<id_movie:bigint,genre:string> +- *(3) Project [user#0L, id_movie#1L, rating#2, timestamp#3L] +- *(3) Filter isnotnull(id_movie#1L) +- *(3) FileScan parquet [user#0L,id_movie#1L,rating#2,timestamp#3L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/ratings.parquet], PartitionFilters: [], PushedFilters: [IsNotNull(id_movie)], ReadSchema: struct<user:bigint,id_movie:bigint,rating:double,timestamp:bigint>

Σχόλια

Ο χρόνος εκτέλεσης χωρις τον optimizer είναι πολύ μεγαλύτερος όπως αναμέναμε διότι το σχήμα του join που ακολουθεί είναι το sortmergejoin. Το συγκεκριμένο σχήμα είναι κατάλληλο όταν έχουμε 2 μεγάλα αρχεία.Εδώ έχουμε ενα μεγάλο και ένα μικρό. Η καθυστέρηση οφείλεται στο γεγονός πως πριν από το join τα δεδομένα ταξινομούνται.