
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ


# Προσαρμοστική Βελτιστοποίηση Εκτέλεσης Ερωτημάτων σε Κατανεμημένα Συστήματα Ροών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ


**Αθανάσιος Δαγκλής**

**Επιβλέπων :** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.



Αθήνα, Σεπτέμβριος 2017


ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ


# Προσαρμοστική Βελτιστοποίηση Εκτέλεσης Ερωτημάτων σε Κατανεμημένα Συστήματα Ροών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ


**Αθανάσιος Δαγκλής**

**Επιβλέπων :** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.


Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Σεπτεμβρίου 2017.

| Νεκτάριος Κοζύρης | Νικόλαος Παπασπύρου | Γεώργιος Γκούμας |
|---|---|---|
| Καθηγητής Ε.Μ.Π. | Αν. Καθηγητής Ε.Μ.Π. | Επ. Καθηγητής Ε.Μ.Π. |

Αθήνα, Σεπτέμβριος 2017

.....

**Αθανάσιος Δαγκλής**

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# Περίληψη

Τα τελευταία χρόνια έχει παρατηρηθεί στροφή προς συστήματα επεξεργασίας δεδομένων ροών, τόσο για ανάλυση μη πεπερασμένων όσο και πεπερασμένων δεδομένων. Η ανάγκη για απαντήσεις σε πραγματικό χρόνο, οι διαρκώς αυξανόμενες πηγές ροών δεδομένων και η ραγδαία αύξηση του μεγέθους των δεδομένων προς επεξεργασία καθιστά την αποδοτική εκτέλεση ερωτημάτων αναγκαία.

Στην παρούσα διπλωματική εργασία, ασχολούμαστε με την βελτιστοποίηση πλάνου εκτέλεσης ερωτημάτων που πραγματοποιούν συνένωση ροών δεδομένων με εξωτερικό σύνολο δεδομένων. Συγκεκριμένα, μελετάμε ερωτήματα με δύο ροές δεδομένων και πολλαπλές ενώσεις με εξωτερικούς πίνακες.

Για να επιτύχουμε προσαρμοστική βελτιστοποίηση πλάνου εκτέλεσης ερωτημάτων, εκτελούμε τα ερωτήματα σε πραγματικό χρόνο χρησιμοποιώντας το Spark Streaming framework. Το σύστημα χρησιμοποιεί τις κατανεμημένες τεχνολογίες Kafka, Spark, HDFS και εξασφαλίζει έτσι την κλιμακωσιμότητα του και την ανοχή σε σφάλματα.

Προκειμένου να επιτευχθεί εφαρμογή μεθόδων βελτιστοποίησης πλάνου εκτέλεσης ερωτημάτων σε πλάνο ροής δεδομένων, εξάγουμε στατιστικά από τις ροές δεδομένων, και τα εγχύουμε στο Spark Streaming. Με αυτό τον τρόπο εφαρμόζεται αποτελεσματικά αναδιάταξη της σειράς των συνενώσεων του πλάνου με βάση πρόβλεψη κόστους και επιλέγονται κατάλληλοι αλγόριθμοι συνένωσης με βάση τα προβλεπόμενα μεγέθη των ενδιάμεσων αποτελεσμάτων.

Επίσης, μελετάμε το είδος και την ακρίβεια των στατιστικών που χρειάζεται να εξάγουμε από τις ροές δεδομένων προκειμένου να έχουμε ικανοποιητική βελτίωση στο χρόνο εκτέλεσης ερωτημάτων.

Εφαρμόζοντας προσαρμοστική βελτιστοποίηση πλάνου εκτέλεσης των ερωτημάτων στις ροές δεδομένων, επιτυγχάνουμε καλύτερους χρόνους εκτέλεσης, με επιτάχυνση έως και 5.5, και κατ επέκτασιν μπορούμε δυναμικά να αποδεσμεύουμε υπολογιστικούς πόρους ώστε αυτοί να αξιοποιούνται από άλλες διεργασίες.



**Λέξεις Κλειδιά:**

Επεξεργασία Ροών Δεδομένων, Προσαρμοστική Βελτιστοποίηση Πλάνου Ερωτημάτων, Βελτιστοποίηση Κόστους, Κατανεμημένα Συστήματα, HDFS, Kafka, Spark Streaming

# Abstract

In recent years, we have increased use of streaming engines both for unbound and bound data. The need for real-time results, the ever increasing unbound data sources and the rapid increase in data sizes make the efficient query execution a necessity.

In this diploma thesis, we are studying the query execution optimization of SQL queries that join streams of data and external datasets. Specifically, we study queries of two streams of data including multiple joins between them and external tables.

To achieve adaptive optimization of query execution, we perform the plans in real time using Spark Streaming framework. The designed system uses distributed technologies such as Kafka, Spark and HDFS, ensuring its scalability and fault tolerance.

In order for the query optimizations of Spark's Query Optimizer, Catalyst, to work, we extract statistics from the streams of data and inject them to Spark Streaming. By doing so, a cost based join reorder optimization is applied to the plan and for each join, the optimizer selects the fitting join algorithm depending on the estimated intermediate result sizes.

Furthermore, we study the kind of required statistics we need to extract from the data streams, as well as their accuracy, in order to achieve satisfactory levels of improvement.

By applying adaptive optimizations of query executions in streaming queries, we are able to complete the queries faster, with a speedup up to 5.5, and dynamically reallocate unused resources to other streaming applications.



**Keywords:**

Stream Processing, Adaptive Optimization of Query Plans, Cost Based Optimizations, Distributed Systems, HDFS, Kafka, Spark Streaming

## Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου Νεκτάριο Κοζύρη, για την δνατότητα που μου έδωσε μέσω αυτού του θέματος να ασχοληθώ σε βάθος με τον τομέα των κατανεμημένων συστημάτων και ανάλυσης ροών δεδομένων.

Θα ήθελα επίσης να ευχαριστήσω τον μεταδιδακτορικό ερευνητή του εργαστηρίου Υπολογιστικών Συστημάτων, Ιωάννη Κωνσταντίνου για την βοήθεια και καθοδήγησή του κατά την διάρκεια της παρούσας εργασίας, καθώς και τον υποψήφιο διδάκτορα Γιάννη Γιαννακόπουλο και τον ερευνητή Βίκτωρα Γιαννακούρη Σαλαλίδη για τις πολλαπλές συζητήσεις και τη συνεχή υποστήριξη σε όσα θέματα αντιμετώπισα.

Τέλος, θέλω να ευχαριστήσω την οικογένειά μου για την υπομονή και τη συνεχή στήριξη που μου παρείχαν καθ όλη τη διάρκεια των σπουδών μου.

Αθανάσιος Δαγκλής

Αθήνα, 12η Σεπτεμβρίου 2017

# Πίνακας περιεχομένων

**ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ**

**ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ**

# 1

# *Εισαγωγή*

## *1.1 Κίνητρο*

Στο πεδίο ανάλυσης δεδομένων, παρατηρείται τα τελευταία χρόνια έντονη δραστηριότητα στον κλάδο ανάλυσης ροών δεδομένων. Η ανάλυση streaming data, γνωστή και ως ανάλυση μη πεπερασμένων/απεριόριστων δεδομένων, παρουσιάζει αυξημένο ενδιαφέρον λόγω των πολλαπλών πλεονεκτημάτων που προσφέρει. Μερικοί λόγοι της αυξημένης δημοτικότητας της ανάλυσης ροών δεδομένων είναι οι εξής:

- Προσφέρει αποτελέσματα πραγματικού χρόνου. Σε μια επιχείρηση, αυτό μπορεί να μειώσει δραματικά το χρόνο απόκρισης και να βοηθήσει στην άμεση λήψη επιχειρηματικών αποφάσεων.
- Η άμεση ανάλυση των δεδομένων, καθώς αυτά έρχονται, βοηθάει σε εξομάλυνση του υπολογιστικού φόρτου και σε πιο προβλέψιμα επίπεδα κατανάλωσης πόρων
- Οι πηγές unbounded δεδομένων διαρκώς αυξάνονται. Δεδομένα από συσκευές IoT, αισθητήρες, analytics εφαρμογών, logs συστημάτων έως και δικτυακή κίνηση προσφέρονται για ανάλυση με συστήματα streaming processing, καθώς είναι ειδικά σχεδιασμένα ώστε να χειρίζονται μη πεπερασμένα δεδομένα.

Τα τελευταία χρόνια, προκειμένου να επιτευχθεί η ανάλυση πραγματικού χρόνου, έχουν δημιουργηθεί νέες υπηρεσίες και έχουν αναπτυχθεί νέες τεχνολογίες και frameworks, με πιο

γνωστά τα εξής:, Apache Storm, Twitter Heron, Apache Flume, Apache Spark, MillWheel, Cloud Dataflow, Apache Flink.

Η αρχική προσέγγιση ανάλυσης πραγματικού χρόνου ήταν η παροχή άμεσης προσεγγιστικής απάντησης ενώ παράλληλα υπολογιζόταν με batch processing και η ακριβής λύση (Lambda architecture). Στη συνέχεια, το Spark Streaming καθιέρωσε τη συνέπεια αποτελεσμάτων και στις αναλύσεις ροών δεδομένων. Αυτό είχε ως αποτέλεσμα τη στροφή προς σωστά σχεδιασμένες μηχανές ανάλυσης μη πεπερασμένων δεδομένων, καθώς έχουν τη δυνατότητα να προσφέρουν και αποτελέσματα και σε πραγματικό χρόνο και εγγυημένα ακριβή.

Υπάρχει πλέον η τάση χρήσης μηχανών ανάλυσης ροών δεδομένων για επεξεργασία τόσο μη πεπερασμένων, όσο και πεπερασμένων δεδομένων. Το Apache Beam είναι το επικρατέστερο προγραμματιστικό μοντέλο που επιχειρεί, κάτω από ένα ενιαίο API, να προσδιορίσει και να εκτελεί ανάλυση πεπερασμένων και μη δεδομένων, με χρήση μηχανών ανάλυσης ροών δεδομένων.

Η διαρκώς αυξανόμενη δημοτικότητα και η έντονη ανάγκη για αποτελέσματα σε πραγματικό χρόνο έχει ως συνέπεια οι απαιτήσεις από τις μηχανές ανάλυσης ροών δεδομένων να αυξάνονται. Η επίλυση των όποιων προβλημάτων αυτές αντιμετωπίζουν γίνεται επιτακτική. Γνωστος περιορισμός τη δεδομένη στιγμή είναι η βελτιστοποίηση σε ερωτήματα ροών. Οι ροές δεδομένων από τη φύση τους είναι ευμετάβλητες στα χαρακτηριστικά τους και καθιστούν την κατάστρωση αποδοτικού πλάνου εκτέλεσης δύσκολη. Όσο αυξάνεται όμως ο όγκος των δεδομένων, τόσο μεγαλύτερο αντίκτυπο έχει η εκτέλεση μη αποδοτικού πλάνου ερωτημάτων.

Τη δεδομένη στιγμή, έχουν γίνει απόπειρες αντιμετώπισης του θέματος της βελτιστοποίησης ερωτημάτων ροών. Συστήματα όπως το Flink, επικεντρώνονται πρωταρχικά σε ροές δεδομένων και ερωτήματα επί αυτών, αλλά δυστυχώς δεν υποστηρίζουν πλήρως ερωτήματα SQL. Συστήματα όπως το Spark και το Storm εφαρμόζουν τεχνικές βελτιστοποίησης ερωτημάτων, όπως θα έκαναν σε batch ερωτήματα όμως, αγνοώντας τα χαρακτηριστικά της ροής δεδομένων. Αυτό έχει ως αποτέλεσμα για ένα ερώτημα να παράγεται διαρκώς το ίδιο πλάνο. Κάτι τέτοιο προφανώς δεν αρκεί, καθώς τα χαρακτηριστικά της ροής δεδομένων μεταβάλλονται με το χρόνο.

## *1.2 Αντικείμενο και συνεισφορά διπλωματικής*

Σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και υλοποίηση συστήματος συνεχούς βελτιστοποίησης SQL ερωτημάτων που περιλαμβάνουν συνένωση ροών δεδομένων και εξωτερικών συνόλων δεδομένων. Με βάση το Spark Streaming, σκοπεύουμε αναλύοντας διαρκώς τις εισόδους ροών δεδομένων και εκμεταλλευόμενοι τον βελτιστοποιητή

ερωτημάτων του Spark SQL, Catalyst, να επιτύχουμε προσαρμοστική κατάστρωση πλάνου εκτέλεσης ερωτημάτων.

Πρωταρχικός στόχος είναι η ελαχιστοποίηση του χρόνου εκτέλεσης των ερωτημάτων με τη συνεχή προσαρμογή του πλάνου στα διαρκώς μεταβαλλόμενα χαρακτηριστικά των ροών δεδομένων. Η ελαχιστοποίηση του χρόνου εκτέλεσης του πλάνου μπορεί στη συνέχεια να οδηγήσει σε απελευθέρωση αχρησιμοποίητων υπολογιστικών πόρων και στην συνολικά αποδοτικότερη εκμετάλλευση των πόρων μας.

Επιμέρους στόχος είναι ο προσδιορισμός των χαρακτηριστικών της ροής που απαιτούνται για επίτευξη ικανοποιητικής βελτίωσης καθώς και ο προσδιορισμός της επίδρασης της ακρίβειας των στατιστικών στο βαθμό βελτίωσης.

Οι κυριότερες συνεισφορές της διπλωματικής συνοψίζονται ως εξής:

- Διαρκής βελτιστοποίηση ερωτημάτων ροών. Εκμεταλλευόμενοι τα στατιστικά των ροών που εξάγουμε και τον βελτιστοποιητή του Spark SQL, Catalyst, επιτυγχάνουμε την συνεχή παραγωγή πλάνου που ανταποκρίνεται στα διαρκώς μεταβαλλόμενα χαρακτηριστικά των εισόδων μας
- Προσδιορισμός απαιτούμενων χαρακτηριστικών ροής για αποτελεσματική βελτιστοποίηση. Μέσω πειραμάτων, αξιολογούμε την αποτελεσματικότητα των στατιστικών και της ακρίβειάς τους στην βελτίωση των παραγώμενων πλάνων εκτέλεσης.

## *1.3 Συσχετιζόμενες εργασίες*

Τα τελευταία χρόνια έχουν προταθεί και αναπτυχθεί διάφορες μέθοδοι βελτιστοποιήσεων πλάνου εκτέλεσης, τόσο για συστήματα ανάλυσης ροών δεδομένων, όσο και για κλασικά συστήματα βάσεων δεδομένων. Ακολουθούν μερικές από τις εργασίες που συσχετίζονται με την παρούσα διπλωματική εργασία.

- **Eddies: Continuously adaptive query processing** [24]. Προτείνεται η εισαγωγή ενός μηχανισμού, του eddy, ο οποίος επιτρέπει την συνεχή προσαρμογή του πλάνου προς εκτέλεση ανά πλειάδα, με χρήση ιστορικών στατιστικών και αναδιάταξη τελεστών,οδηγώντας σε δραματική βελτίωση σε δυναμικά περιβάλλοντα εκτέλεσης. Αυτή η εργασία αποτέλεσε την κύρια έμπνευση για την διπλωματική εργασία, καθώς προτείνει την συνεχή βελτίωση του πλάνου εκτέλεσης του ερωτήματος κατά την εκτέλεσή του. Στην περίπτωσή μας, αντί να αναδιατάσσουμε τελεστές ανά πλειάδα, επαναξιολογούμε το πλάνο ανά κάθε παρτίδα (batch). Σε πρώτη φάση, η διάταξη προς αξιολόγηση σχεδιάστηκε ακολουθώντας πιστά τη λογική των Eddies και χρησιμοποιούσε ιστορικά στατιστικά, ενώ στη συνέχεια εκμεταλλευτήκαμε τη

δυνατότητα παροχής των πραγματικών στατιστικών για επίτευξη καλύτερων αποτελεσμάτων.

- **Rate-Based Query Optimization for Streaming Information Sources** [25]. Προτείνεται η χρήση ρυθμού εισόδου των ροών δεδομένων από τους βελτιστοποιητές ερωτημάτων, ως αντικαταστάτης της πληθικότητας με στόχο τη μεγιστοποίηση του ρυθμού εξόδου. Στη περίπτωσή μας κάτι τέτοιο δεν μπορεί να εφαρμοστεί, λόγω του τρόπου λειτουργίας του Spark Streaming με micro batching, όπου δεν υπάρχει η έννοια του ρυθμού εισόδου των ροών δεδομένων.
- **Spark's Cost Based Optimizations** [19, 20]**.** Μια από τις τελευταίες προσθήκες στον βελτιστοποιητή ερωτημάτων του Spark, Catalyst. Με χρήση στατιστικών επί των στατικών δεδομένων επιτυγχάνεται εφαρμογή βελτιστοποιήσεων κόστους στο λογικό πλάνο εκτέλεσης ερωτημάτων. Οι περιορισμοί του είναι πως προς το παρόν λειτουργεί μόνο με δεδομένα από Hive, τα οποία παρέχουν τα απαιτούμενα στατιστικά και πως δεν λειτουργεί σε ανάλυση ροών δεδομένων από Spark Streaming. Παρόμοιες βελτιστοποιήσεις κόστους έχουν γίνει απευθείας στο Hive, αλλά αφορούν μόνο εκτέλεση ερωτημάτων σε στατικά δεδομένα και όχι σε ροές.
- **Strider: A Hybrid Adaptive Distributed RDF Stream Processing Engine** [31]. Υλοποιεί μια υβριδική μηχανή επεξεργασίας ροών δεδομένων, η οποία βασίζεται στα Spark και Kafka και αξιοποιώντας διαρκώς την κατάσταση των ροών δεδομένων κατά την βελτιστοποίηση πλάνου επιτυγχάνει έως και 60x υψηλότερη απόδοση από αντίστοιχα συστήματα. Μειονεκτήματα του Strider είναι ο περιορισμός των ερωτημάτων που μπορεί να τρέξει, καθώς τρέχει ερωτήματα SPARKQL.

## *1.4 Οργάνωση κειμένου*

Στο Κεφάλαιο 2 παρουσιάζεται το γενικό θεωρητικό υπόβαθρο που απαιτείται για την πλήρη κατανόηση της διπλωματικής. Περιλαμβάνει πληροφορίες για τις ροές δεδομένων, τους βελτιστοποιητές ερωτημάτων, τα frameworks και τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

Στο Κεφάλαιο 3 παρουσιάζεται η υλοποίηση του συστήματος. Αρχικά αναλύεται ο τρόπος εξαγωγής στατιστικών από τις εισόδους ροών δεδομένων. Στη συνέχεια, περιγράφεται ο τρόπος έγχυσης στατιστικών στα DataFrames των ερωτημάτων και τέλος περιγράφεται η εφαρμογή Spark Streaming, η οποία εκμεταλλευόμενη τα στατιστικά που εξάγονται και αξιοποιώντας την έγχυση στατιστικών, που εκτελεί τα ερωτήματα επιτυγχάνοντας προσαρμοστική βελτιστοποίηση του πλάνου.

Στο Κεφάλαιο 4 παρουσιάζεται η πειραματική αξιολόγηση του συστήματος. Αρχικά, περιγράφεται η πειραματική διάταξη στην οποία έτρεξαν τα πειράματα. Στη συνέχεια, παρουσιάζονται το dataset και τα ερωτήματα επί αυτού, τα κριτήρια αξιολόγησης των πειραμάτων, καθώς και ο λόγος επιλογής τους. Τέλος, παρατίθενται τα σενάρια των πειραμάτων, τα αποτελέσματά τους και η αξιολόγηση των αποτελεσμάτων.

Στο Κεφάλαιο 5 παρουσιάζονται τα τελικά συμπεράσματα της εργασίας και προτείνονται μελλοντικές επεκτάσεις της.

# 2

# *Θεωρητικό Υπόβαθρο*

## *2.1 Ροές Δεδομένων*

Τα τελευταία χρόνια εμφανίζεται διαρκώς ανάγκη ανάλυσης δεδομένων που παράγονται από πολλαπλές πηγές, όπως από κίνηση δικτύων, διαδικτυακές εφαρμογές, αισθητήρες και IoT συσκευες. Χαρακτηριστικά των δεδομένων αυτών είναι ο υψηλός ρυθμός παραγωγής τους και ο μεγάλος τους όγκος. Με τη συνεχή αύξηση των πηγών δεδομένων, αυξάνεται διαρκώς και ο όγκος των παραγόμενων δεδομένων, σε βαθμό που δεν μπορούν να αναλυθούν με παραδοσιακό τρόπο από κλασικά συστήματα ανάλυσης δεδομένων και η μόνιμη αποθήκευσή τους καθίσταται αδύνατη, λόγω του απαγορευτικού κόστους με τον όγκο των δεδομένων διαρκώς να αυξάνεται.

Προκειμένου να καταστεί δυνατή η ανάλυση τέτοιων δεδομένων, τα αντιμετωπίζουμε ως ροές δεδομένων. Αποφεύγουμε την αποθήκευση του πλήρους όγκου των δεδομένων, επεξεργαζόμαστε διαρκώς τμήματα του συνόλου και απορρίπτουμε παλαιότερα δεδομένα. Λόγω της συνεχούς προσθήκης νέων δεδομένων, οι ροές δεδομένων μπορούν να χαρακτηριστούν και απεριόριστα σύνολα δεδομένων, σε αντίθεση με τα πεπερασμένα δεδομένα τα οποία και επεξεργάζονται τα κλασικά συστήματα επεξεργασίας δεδομένων. Ένα σωστά σχεδασμένο σύστημα επεξεργασίας απεριόριστων δεδομένων αποτελεί υπερσύνολο των συστημάτων επεξεργασίας δεδομένων και μπορεί να επεξεργαστεί και πεπερασμένα σύνολα δεδομένων.

Το κυριότερο χαρακτηριστικό των ροών δεδομένων που θα μας απασχολήσει στα πλαίσια της διπλωματικής εργασίας είναι η απρόβλεπτη φύση τους. Σε αντίθεση με τα συμβατικά συστήματα ανάλυσης δεδομένων, όπου το σύνολο των δεδομένων υπάρχει στο σύστημα πριν την εκτέλεση ερωτημάτων και μπορεί να αναλυθεί ώστε να επιτευχθεί η βέλτιστη εκτέλεση τυ ερωτήματος, στην ανάλυση ροών, τα δεδομένα αλλάζουν συνεχώς καθώς διαρκώς καταφθάνουν νέα δεδομένα, επομένως δεν έχουμε γνώση της μορφής τους πριν την εκτέλεση του ερωτήματος. Αυτό έχει ως αποτέλεσμα την αδυναμία συστημάτων επεξεργασίας ροών δεδομένων να λάβουν υπόψιν τους τον διαρκώς μεταβαλλόμενο ρυθμό των ροών δεδομένων και την συνεχή αλλαγή των χαρακτηριστικών τους.

Οι υπάρχουσες προσεγγίσεις για βελτιστοποιήσεις ερωτημάτων σε ροές δεδομένων εστιάζουν σε εφαρμογή ευρεστικών και λογικών κανόνων, αγνοώντας όμως τις παραπάνω ιδιαιτερότητες των ροών.

## *2.2 Kafka*

Το Kafka είναι μια κατανεμημένη πλατφόρμα streaming που επιτρέπει την δημοσίευση μηνυμάτων σε ροές δεδομένων, την εγγραφή σε συγκεκριμένες ροές δεδομένων, όπως μια ουρά μηνυμάτων, την αποθήκευση των εγγραφών των ροών δεδομένων ώστε να εξασφαλίζεται η ανοχή σε σφάλματα και την επεξεργασία ροών δεδομένων σε πραγματικό χρόνο.

Οι θεμελιώδεις οντότητες του Kafka είναι οι ροές εγγραφών, οι οποίες αποθηκεύονται ανά κατηγορίες γνωστές ως θέματα – topics. Κάθε εγγραφή αποτελείται από ένα κλειδί, μια τιμή και ένα timestamp. Ανάλογα με το κλειδί της, η εγγραφή μπορεί να κατανεμηθεί περαιτέρω σε ξεχωριστό partition του topic, το οποίο κατ επέκτασιν μπορεί να σημαίνει ξεχωριστό κόμβο Kafka.

Με τον server που τρέχει Kafka επικοινωνούν μέσω 4 APIs οι εξής:

- Παραγωγοί: δημοσιεύουν δεδομένα στις ροές εγγραφών ενός ή περισσότερων Kafka topics.
- Καταναλωτές: εγγράφονται σε Kafka topics και στη συνέχεια λαμβάνουν και επεξεργάζονται όπως επιθυμούν τα μηνύματα των ροών της επιλογής τους.
- Stream Processors: εφαρμογές επεξεργασίας ροών. Δέχονται ροές δεδομένων ως είσοδο και παράγουν ροές δεδομένων ως έξοδο. Stream Processor είναι και το Spark Streaming.
- Connectors: δημιουργούν επαναχρησιμοποιήσιμους παραγωγούς και καταναλωτές που συνδέουν Kafka topics σε υπάρχουσες εφαρμογές ή συστήματα.

Κατά την εγγραφή πολλαπλών μηνυμάτων σε Kafka server από έναν παραγωγό, έχουμε την εγγύηση πως η σειρά των μηνυμάτων του ίδιου παραγωγού παραμένει σταθερή στην ροή εγγραφών που δημιουργείται και συνεπώς θα αναγνωστεί με τη σωστή σειρά από τους καταναλωτές.

Η ανάγνωση ενός Kafka topic γίνεται με χρήση των offsets. Ο καταναλωτής αρκεί να γνωρίζει μόνο το offset της τελευταίας του ανάγνωσης και το Kafka του εγγυάται πως δε θα διπλοδιαβάσει ούτε θα χάσει κάποιο μήνυμα και πως θα διαβάσει με τη σωστή σειρά όλα τα μηνύματα.

## *2.3 Spark*

### *2.3.1 Εισαγωγικές έννοιες*

Το Apache Spark είναι μια γρήγορη, γενικού σκοπού υπολογιστική πλατφόρμα ανοιχτού λογισμικού, για επεξεργασία δεδομένων μεγάλης κλίμακας. Είναι ειδικά σχεδιασμένο για cluster υπολογιστών και εξασφαλίζει αυτόματα παράλληλη επεξεργασία δεδομένων, κλιμακωσικότητα και ανοχή σε σφάλματα.

Το Spark αποτελεί μια προέκταση του μοντέλου MapReduce, σχεδιασμένη έτσι ώστε πέρα από batch processing να υποστηρίζει και διαδραστικά ερωτήματα (interactive queries), επεξεργασία ροών δεδομένων (streaming queries) και machine learning.

Η αρχιτεκτονική του Spark φαίνεται στο παρακάτω σχήμα.

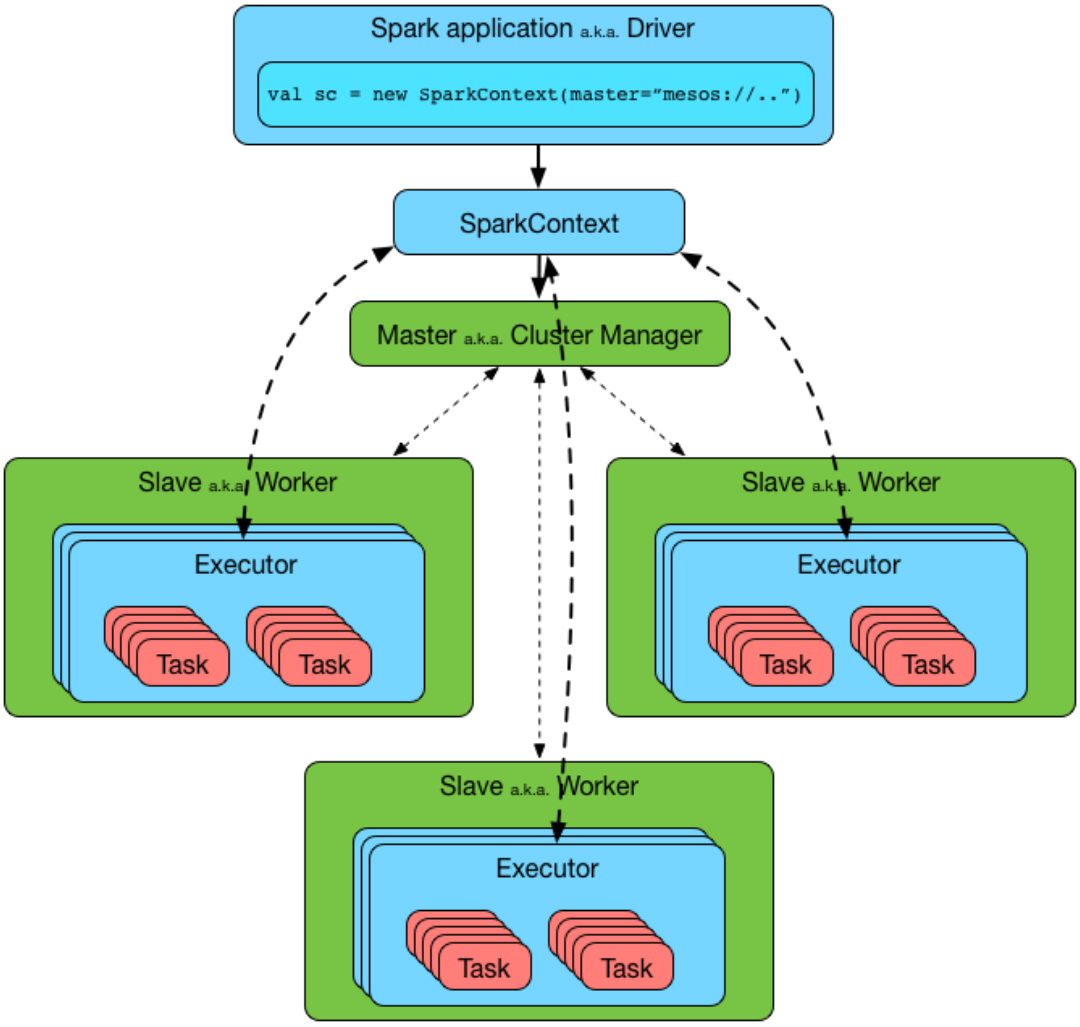


**Figure 2-1 Αρχιτεκτονική Spark**

Δομική μονάδα του Spark είναι το Resilient Distributed Dataset (RDD). Το RDD είναι μια συλλογή αντικειμένων δεδομένων μόνο προς ανάγνωση, κατανεμημένο σε ένα cluster υπολογιστών με τη δυνατότητα ανάκτησης σε περίπτωση απώλειας κάποιου τμήματος, ώστε να εξασφαλίζεται η ανοχή σε σφάλματα. Είναι σχεδιασμένο ώστε να τρέχει σε Hadoop cluster και έχει τη δυνατότητα να χρησιμοποιεί δεδομένα από HDFS, HBase, Casssandra, Hive και οποιοδήποτε άλλο Hadoop InputFormat.

Το θεμέλιο του Spark είναι το Spark Core. Αποτελεί την γενική μηχανή εκτέλεσης για την πλατφόρμα του Spark. Παρέχει υπολογισμούς In-Memory, βασικές λειτουργικότητες I/O και κατανεμημένο διαμοιρασμό και χρονοδρομολόγηση των tasks.

Πάνω στη βάση του Spark Core, έχουν χτιστεί βιβλιοθήκες εκτέλεσης SQL και DataFrames ερωτημάτων (Spark SQL), επεξεργασίας ροών δεδομένων (Spark Streaming), γράφων (GraphX) και machine learning (MLlind).

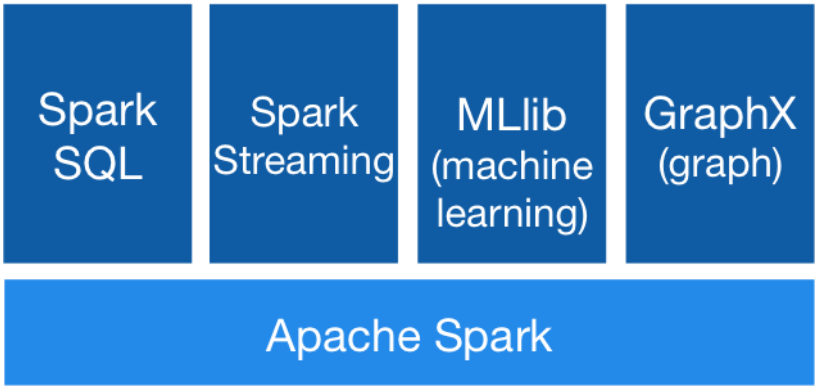


**Figure 2-2 Κεντρικές Βιβλιοθήκες Spark Project**

Στην παρούσα διπλωματική θα ασχοληθούμε με το Spark SQL και το Spark Streaming.

### *2.3.2 Spark SQL*

Η βιβλιοθήκη Spark SQL είναι το τμήμα του Spark για την επεξεργασία δομημένων δεδομένων. Εισάγει την αφαιρετική έννοια των DataFrames, τα οποία παρέχουν υποστήριξη για δομημένα και ημι-δομημένα δεδομένα. Ξεπερνώντας το βασικό API των Spark RDDs, η διεπαφή που προσφέρει το Spark SQL παρέχει περισσότερη πληροφορία για τη δομή τόσο των δεδομένων, όσο και των υπολογισμών που εκτελούνται. Εσωτερικά, η επιπρόσθετη αυτή πληροφορία αξιοποιείται από τον βελτιστοποιητή του Spark SQL, Catalyst, για την επίτευξη επιπρόσθετων βελτιστοποιήσεων.

Δυνατότητα χρήσης του Spark SQL δίνεται είτε μέσω SQL είτε μέσω του DataFrame API.

### *2.3.3 Catalyst*

Ο Catalyst είναι ο βελτιστοποιητής ερωτημάτων του Spark SQL. Χρησιμοποιείται τόσο σε SQL ερωτήματα, όσο και σε ερωτήματα με χρήση του DataFrame API. Είναι γραμμένος σε Scala και έχει φτιαχτεί ώστε να είναι εύκολα επεκτάσιμος.

#### *2.3.3.1 Τρόπος λειτουργίας*

Ο Catalyst δέχεται ως είσοδο ένα πλάνο είτε σε SQL, είτε γραμμένο σε DataFrames και παράγει κατόπιν ανάλυσης και εφαρμογής κανόνων ως έξοδο ένα βέλτιστο πλάνο.

Αρχικά, το πλάνο εισόδου από σκέτο κείμενο SQL περνάει από αναλυτή SQL ώστε να παραχθεί ένα ασαφές συνταντικό δέντρο. Σε αυτή τη φάση γίνεται απλή συνταντική ανάλυση του ερωτήματος και όχι έλεγχος εγκυρότητάς του και ύπαρξης των πινάκων και στηλών που αφορά.

Στη συνέχεια, με βοήθεια του Catalog αντικειμένου που διατηρεί το Spark, γίνεται σταδιακά αποτίμηση του μη αποτιμημένου λογικού πλάνου που δημιουργήθηκε και αντιστοιχίζονται τα

ονόματα πινάκων με πίνακες, οι ιδιότητες των πινάκων με στήλες και οι τύποι των αποτελεσμάτων των εκφράσεων του ερωτήματος, ώστε να διευκολυνθεί η εφαρμογή κανόνων βελτιστοποίησης σε επόμενα βήματα.

Αφού παραχθεί ένα λογικό πλάνο εκτέλεσης ερωτήματος, ακολουθεί το στάδιο λογικών βελτιστοποιήσεων, το στάδιο βελτιστοποιήσεων κόστους και τέλος παράγεται το φυσικό πλάνο και ο ο κώδικας εκτέλεσης του βελτιστοποιημένου πλάνου.

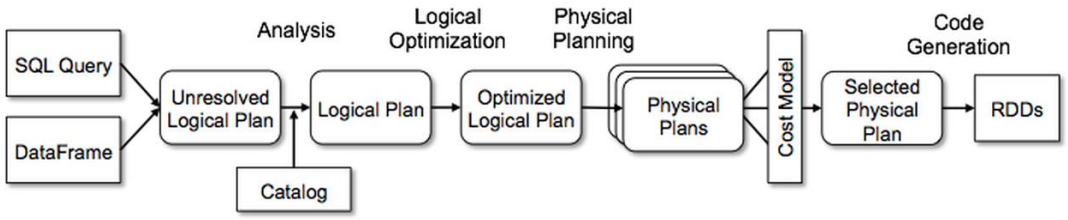


**Figure 2-3 Σταδια λειτουργίας Catalyst**

Ακολουθεί ανάλυση των επιπέδων βελτιστοποίησης του Catalyst.

### *2.3.3.2 Κανόνες λογικής βελτιστοποίησης*

Οι λογικές βελτιστοποιήσεις στον Catalyst επιτυγχάνονται με διαδοχική εφαρμογή λογικών κανόνων στο δέντρο του λογικού πλάνου. Οι κανόνες εφαρμόζονται διαδοχικά με αναγνώριση μοτίβων στο δέντρο. Οι λογικοί κανόνες έχουν ως αποτέλεσμα την συνένωση κόμβων με πράξεις σταθερών και απαλοιφή περιττών κόμβων, την διάδοση κενών σχέσεων, την απαλοιφή aliases και περιττών προβολών, την απλοποίηση φίλτρων, λογικών συνθηκών, κ.α. και την προώθηση Filter και Project τελεστών όσο πιο κοντά στα φύλλα του δέντρου γίνεται. Στις φιγούρες 2.4, 2.5 και 2.6 παρατηρούμε παραδείγματα διαδοχικής εκτέλεσης λογικών κανόνων βελτιστοποίησης και πως αυτοί αλλάζουν το δέντρο του ερωτήματός μας.

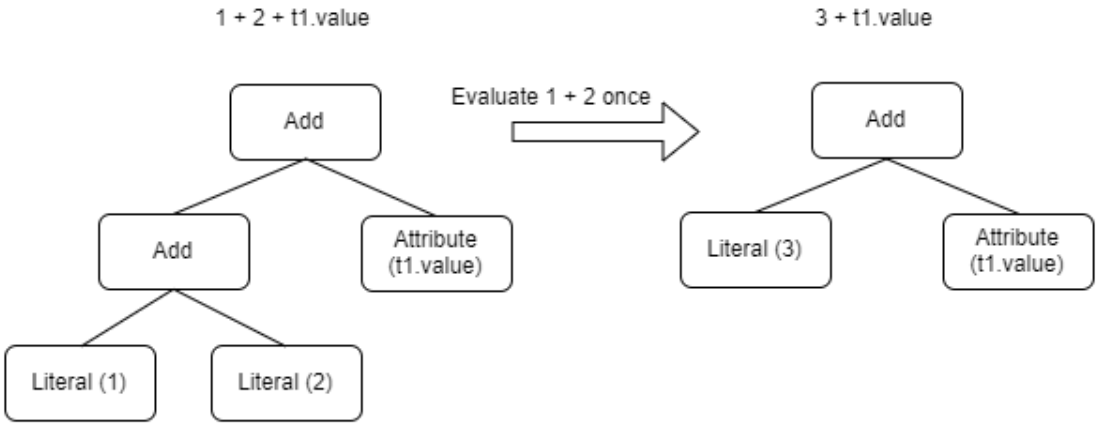


**Figure 2-4 Αποτίμηση πράξεων σταθερών**

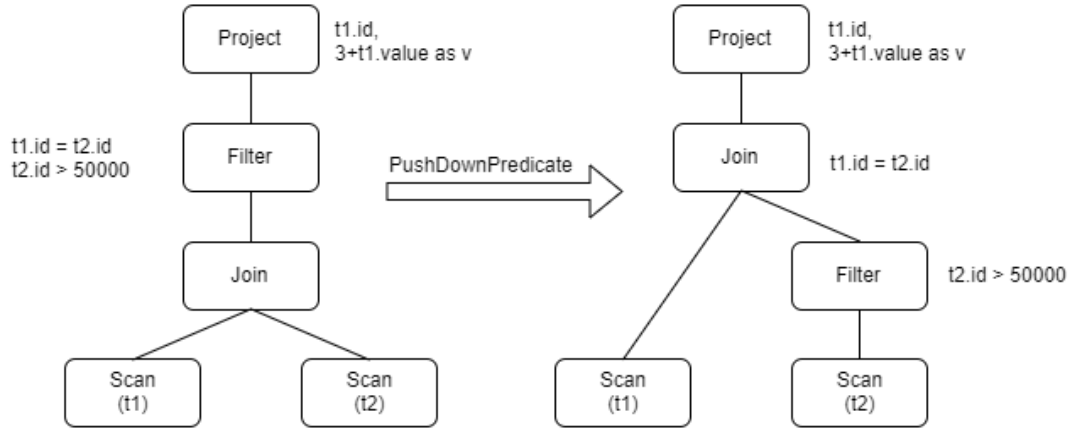


**Figure 2-5 Προώθηση φίλτρου**

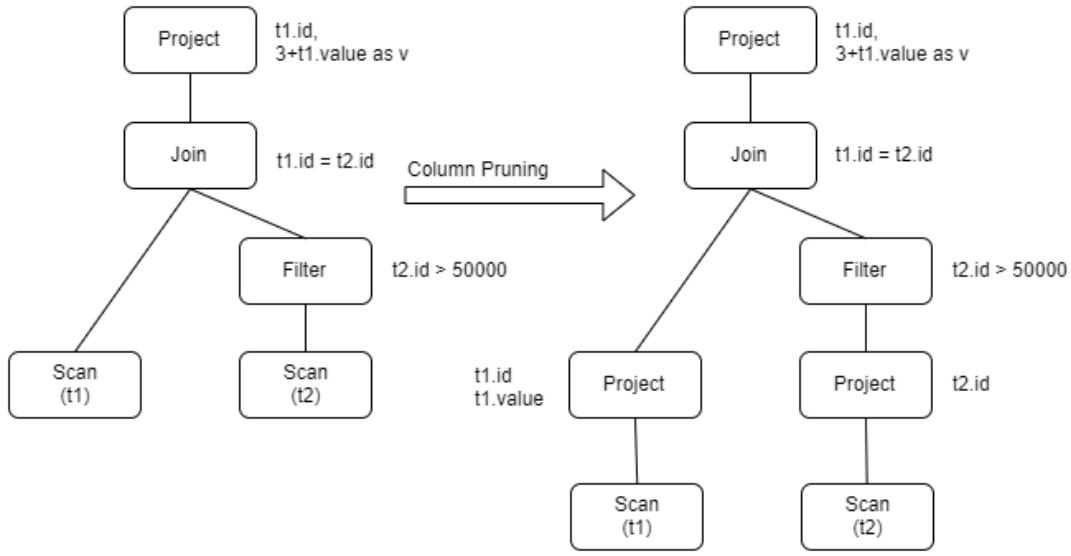


**Figure 2-6 Κλάδεμα στηλών**

Με την εφαρμογή των λογικών κανόνων επιτυγχάνεται η δημιουργία νέου πλάνου, το οποίο είναι πλέον απαλαγμένο από περριτά στάδια, αναδιατεταγμένο και εμπλουτισμένο ώστε να γίνεται το συντομότερο δυνατόν κλάδεμα κόμβων και στηλών των πινάκων ώστε να ελαχιστοποιηθεί η μεταφορά δεδομένων και ο χρόνος εκτέλεσης του ερωτήματος.

Το πλάνο λογικών βελτιστοποιήσεων είναι σαφώς ανώτερο από το απλό λογικό πλάνο. Ο ένας μεγάλος του περιορισμός είναι πως όλες οι βελτιστοποιήσεις που εφαρμόζονται βασίζονται σε κανόνες, αγνοώντας πλήρως τη μορφή των δεδομένων που εμπλέκονται στο πλάνο.Γι αυτόν το λόγο, όταν υπάρχουν στοιχεία για τα χαρακτηριστικά των δεδομένων του ερωτήματος, τις βελτιστοποιήσεις λογικών κανόνων ακολουθούν οι βελτιστοποιήσεις κόστους.

#### *2.3.3.3 Κανόνες βελτιστοποίησης κόστους*

Στην έκδοση του Spark (2.2.0 – 11/7/2017) που χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία, οι βελτιστοποιήσεις βελτιστοποίησης κόστους του Catalyst περιορίζονται σε 2 βασικές.

Η πρώτη και παλαιότερη βελτιστοποίηση είναι η επιλογή αλγορίθμου συνένωσης με βάση το μέγεθος των πινάκων που συνενώνονται. Ο αλγόριθμος που χρησιμοποιείται από το Spark για συνενώσεις πινάκων είναι ο Sort Merge Join, ο οποίος έχει ως προαπαιτούμενο την ταξινόμηση και των δύο πινάκων προς συνένωση. Αν έστω και ένας εκ των δύο πινάκων έχει μέγεθος μικρότερο ενός ορίου (autoBroadcastJoinThreshold), επιλέγεται ως αλγόριθμος συνένωσης ο αλγόριθμος Broadcast Join, ο οποίος επιλέγει τον μικρότερο εκ των δύο πινάκων, τον κάνει hash και στη συνέχεια broadcast προς όλους τους executors. Πρακτικά, αυτό σημαίνει πως αν έχουμε συνένωση ενός πίνακα 100MB με έναν πίνακα 50GB, επιλέγοντας Broadcast Join αρκεί να αποστείλουμε σε όλους τους Workers 100MB και αποφεύγουμε την ταξινόμηση του μεγάλου πίνακα.

Το όριο αυτόματης επιλογής αλγορίθμου Broadcast Join δεν απαιτεί ενεργοποίηση των Cost Based Optimizations, όμως για να λειτουργήσει όσο πιο αποδοτικά γίνεται απαιτείται ακριβής πρόβλεψη των μεγεθών των πινάκων, κάτι το οποίο θα αναλυθεί παρακάτω.

Η δεύτερη και πιο πρόσφατη βελτιστοποίηση κόστους είναι η αναδιάταξη της σειράς διαδοχικών συνενώσεων. Η βελτιστοποίηση αυτή έχει ως στόχο την αναδιάταξη των συνενώσεων με τρόπο τέτοιο ώστε το συνολικό κόστος των τελεστών συνένωσης να ελαχιστοποιηθεί. Αυτό επιτυγχάνεται ως εξής: Αρχικά διατρέχουμε το δέντρο του λογικού πλάνου και ανιχνεύουμε διαδοχικές συνενώσεις. Αν οι συνενώσεις είναι περισσότερες απο δύο και κάθε συνένωση έχει στατιστικά για το μέγεθός της, προχωράμε σε αναζήτηση βέλτιστης σειράς. Τη φθηνότερη σειρά εκτέλεσης των συνενώσεων τη βρίσκουμε με δυναμικό προγραμματισμό, υπολογίζοντας σε κάθε επίπεδο το κόστος κάθε συνένωσης και επιλέγοντας το φθηνότερο. Συνεχίζοντας μέχρι τέλους καταλήγουμε στη βέλτιστη σειρά συνενώσεων. Για τη σύγκριση κόστους σε κάθε επίπεδο εισάγεται ξεχωριστό μοντέλο κόστους και ως κριτήρια καλύτερης συνένωσης λαμβάνονται υπόψιν ο λόγος των μεγεθών και ο λόγος της πληθικότητας των δύο πινάκων, βάσει μιας σταθεράς που έχουμε θέσει.

Συγκεκριμένα, ένα πλάνο θεωρείται καλύτερο απο ένα άλλο αν:

```
def betterThan(other: JoinPlan, conf: SQLConf): Boolean = {
  relativeRows = this.planCost.card / other.planCost.card
  relativeSize = this.planCost.size / other.planCost.size
  relativeRows * conf.joinReorderCardWeight + relativeSize * (1 –
    conf.joinReorderCardWeight) < 1
```

}

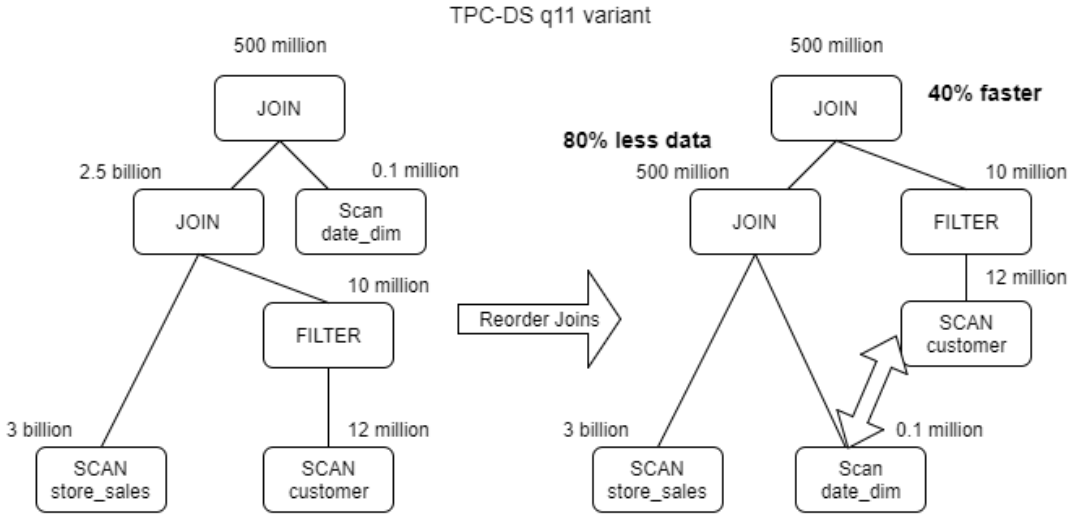


**Figure 2-7 Παράδειγμα αναδιάταξης συνενώσεων**

Πυρήνας των βελτιστοποιήσεων κόστους είναι η ύπαρξη στατιστικών σε κάθε επίπεδο του πλάνου προς βελτιστοποίηση. Αν και τα απαιτούμενα στατιστικά και για τις δυο βελτιστοποιήσεις είναι μόνο το μέγεθος τον πινάκων σε bytes και το πλήθος των γραμμών τους, στην πράξη χρησιμοποιούνται πολύ πιο αναλυτικά στατιστικά για την πρόβλεψη μεγεθών και την εξέλιξή τους μέσα από τους διάφορους τελεστές.

Συγκεκριμένα, τα στατιστικά που αναγνωρίζει και αξιοποιεί το Spark είναι τα εξής:

- sizeInBytes: μέγεθος του πίνακα σε Bytes
- rowCount: εκτιμώμενο πλήθος σειρών του πίνακα
- attributeStats: αναλυτικά στατιστικά για κάθε στήλη του πίνακα που περιλαμβάνουν τα παρακάτω
    - distinctCount: πλήθος των διακριτών τιμών της στήλης
    - min: ελάχιστη τιμή της στήλης
    - max: μέγιστη τιμή της στήλης
    - nullCount: πλήθος null τιμών
    - avgLen: μέσο μήκος τιμών της στήλης
    - maxLen: μέγιστο μήκος τιμών της στήλης

Όταν έχουμε πλήρη στατιστικά και έχουμε ενεργοποιημένες τις βελτιστοποιήσεις κόστους, σε κάθε επίπεδο υπολογίζονται τα στατιστικά του κόμβου αναδρομικά, ξεκινώντας από τα στατιστικά των φύλλων. Ο κάθε τελεστής υλοποιεί την συνάρτηση computeStats, η οποία με βάση το είδος του τελεστή και τα στατιστικά των παιδιών του υπολογίζει τα νέα στατιστικά του κόμβου.

Ένα εύκολο παράδειγμα υπολογισμού στατιστικών είναι του τελεστή filter. Προς το παρόν, ο τελεστής filter θεωρεί πως υπάρχει ομοιόμορφη κατανομή των τιμών μεταξύ της μέγιστης και ελάχιστης τιμής μιας στήλης. Επομένως, ένα φίλτρο της μορφής Column A < value B επηρεάζει ως εξής τα στατιστικά του κόμβου φίλτρου:

- Αν B < A.min, έχουμε filtering factor 0%, επομένως τα στατιστικά πρέπει να αλλάξουν
- Αν A.min < B < A.max, θεωρώντας ομοιόμορφη κατανομή, έχουμε:
    - filtering factor = (B – A.min) / (A.max – A.min)
    - A.min = A.min
    - A.max = B
    - A.distinct = A.distinct* filtering factor
- Αν A.max < B, έχουμε filtering factor 100% και επομένως τα στατιστικά μένουν αμετάβλητα.

Μια άμεση συνέπεια της καλής εκτίμησης στατιστικών παρατηρούμε στο παρακάτω σχήμα, όπου χάρη στην καλή πρόβλεψη στατιστικών φίλτρου επιλέγουμε Broadcast Join αλγόριθμο συνένωσης, αποφεύγοντας έτσι το χρονοβόρο Sort Merge Join που απαιτεί την ταξινόμηση του μεγάλου πίνακα.

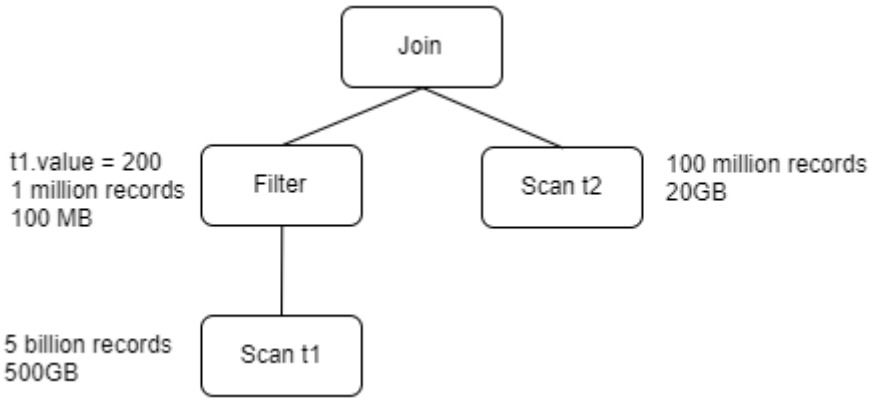


**Figure 2-8 Παράδειγμα χρησιμότητας στατιστικών ακριβείας**

### *2.3.4 Spark Streaming*

Η βιβλιοθήκη Spark Streaming περιέχει την επέκταση του Spark που του επιτρέπει να εκτελεί ερωτήματα και ανάλυση ροών δεδομένων. Χρησιμοποιεί το API υψηλού επιπέδου του Spark, επιτρέποντας έτσι στους χρήστες να γράφουν εφαρμογές ανάλυσης ροών δεδομένων με τον ίδιο τρόπου που θα έγραφαν εφαρμογές ανάλυσης στατικών δεδομένων. Το Spark Streaming παρέχει στις εφαρμογές του κλιμακωσιμότητα, υψηλή απόδοση και ανοχή σε σφάλματα. Ως εισόδους ροών δεδομένων μπορεί να δεχτεί ροές από Kafka, Flume, Kinesis, TCP sockets,

κ.α.. Επιτρέπει την εκτέλεση σύνθετων αλγορίθμων και υψηλού επιπέδου συναρτήσεων επί των ροών καθώς και την επεξεργασία τους με machine learning και graph processing με τις αντίστοιχες βιβλιοθήκες του Spark.

Εσωτερικά, λειτουργεί ως εξής: Λαμβάνει ροές δεδομένων και τις διαμοιράζει σε παρτίδες, τις οποίες και προωθεί στο Spark, προκειμένου να παραχθούν τα αποτελέσματα των παρτίδων όπως φαίνεται στην παρακάτω εικόνα. Ο τρόπος λειτουργίας του είναι γνωστός ως micro-batching.

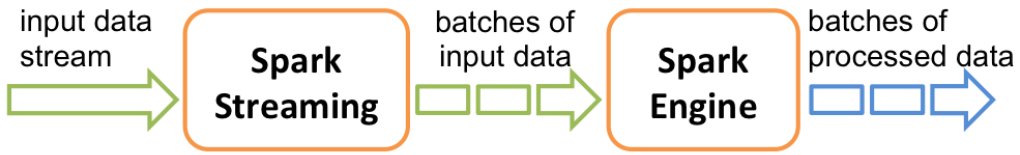


**Figure 2-9 Επεξεργασία ροών δεδομένων με Spark Streaming**

Στους δημιουργούς streaming εφαρμογών, το Spark Streaming προσφέρει την υψηλού επιπέδου αφαιρετική οντότητα των διακριτοποιημένων ροών ή αλλιώς DStreams. Τα DStreams αντιπροσωπεύουν τις ροές δεδομένων που προκύπτουν από πηγές όπως Kafka, Flume, Kinesis ή ακόμα και από εφαρμογή τελεστών υψηλού επιπέδου σε άλλα DStreams. Αφού καθοριστεί το χρονικό διάστημα μιας παρτίδας, καθώς και το χρονικό παράθυρο ανάλυσης, εκκινείται η ανάλυση του DStream και το Spark Streaming φροντίζει αυτόματα ανά τα σταθερά χρονικά διαστήματα που ορίσαμε να προετοιμάζει τις παρτίδες δεδομένων από τις ροές δεδομένων, να δημιουργεί RDDs και να τα αποστέλει προς επεξεργασία στο Spark.

Μια σημαντική συνάρτηση την οποία χρησιμοποιούμε ως πυρήνα της Spark Streaming εφαρμογής μας είναι η foreachRDD. Η foreachRDD είναι από τις πιο γενικές συναρτήσεις του Spark Streaming. Παίρνει ως όρισμα μια συνάρτηση, η οποία και εκτελείται στον Driver του Spark για κάθε παρτίδα δεδομένων που δημιουργεί το Spark Streaming και έχει πρόσβαση στα RDDs των ροών δεδομένων.

### *2.3.5 Spark Streaming Kafka Integration*

Καθώς η ροές δεδομένων που εξετάζουμε προέρχονται από Kafka, αξίζει να επισημάνουμε ορισμένα σημεία της ενσωμάτωσης Kafka στο Spark Streaming. Οι ροές δεδομένων που προέρχονται από Kafka είναι πλήρως συμβατές με το Spark Streaming, καθώς έχει αναπτυχθεί σχετική βιβλιοθήκη για τη βέλτιστη αξιοποίηση των ιδιαιτεροτήτων του Kafka.

Χάρη στην ανοχή στα σφάλματα που έχουν τόσο το Spark Streaming όσο και το Kafka, με την από κοινού χρήση τους επιτυχγάνυμε την κατασκευή συστήματος με υψηλή ανοχή σε

σφάλματα. Μάλιστα, λόγω των offsets των καταναλωτών του Kafka, έχουμε και εγγυημένα μοναδική εκτέλεση ανάλυσης για κάθε εγγραφή, αποφεύγοντας πιθανά σφάλματα πολλαπλής επεξεργασίας.

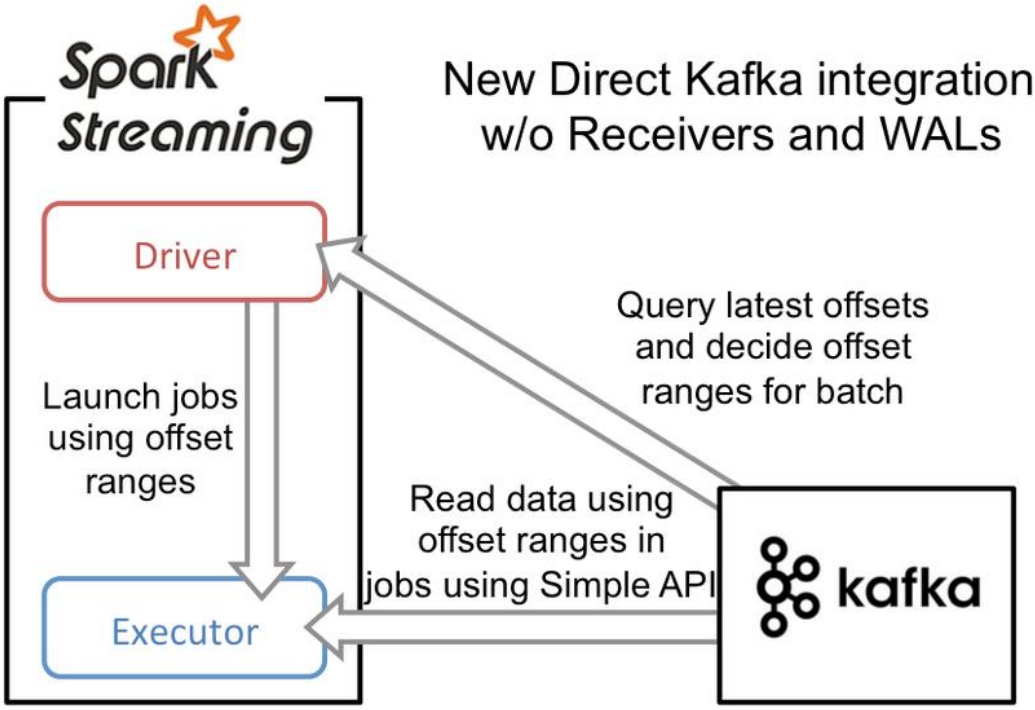


**Figure 2-10 Spark Streaming Kafka Integration**

Από πλευράς μεταφοράς δεδομένων, το Spark Streaming φροντίζει να παρακολουθεί την εξέλιξη της ροής Kafka και όταν ολοκληρωθεί μια παρτίδα, ενημερώνει τον Driver για τα offsets της ροής τα οποία και καθορίζουν την παρτίδα προς ανάλυση. Στη συνέχεια, ο Driver, διαμοιράζει τα partitions του κάθε topic στους executors και τους αποστέλει τα tasks, μαζί με τα offsets που καθορίζουν τα τμήματα της κάθε παρτίδας που θα επεξεργαστεί ο κάθε executor. Έτσι, πρακτικά από τον Driver και την εφαρμογή Spark Streaming περνάει μόνο η πληροφορία των offsets που προσδιορίζουν την κάθε παρτίδα και όχι δεδομένα της ίδιας της ροής.

# 3

# *Υλοποίηση*

## *3.1 Δημιουργία ροών δεδομένων*

Όπως προαναφέρθηκε στον τρόπο λειτουργίας του Spark Streaming, οι ροές δεδομένων εκλαμβάνονται ως ξεχωριστές πεπερασμένες «παρτίδες» δεδομένων. Για να επιτύχουμε αλλαγή των χαρακτηριστικών των ροών δεδομένων, μιας και δεν υπάρχει η έννοια του ρυθμού εισόδου τους στα πλαίσια του Spark Streaming, φροντίζουμε να αυξομειώνουμε τα μεγέθη από τις παρτίδες προς ανάλυση.

Η δημιουργία κάθε ροής είναι αποτέλεσμα μιας διεργασίας python. Η κεντρική συνάρτηση είναι η produce_stream.

```
def produce_stream(
      broker, topic, source, conf_path, timeout):
```

broker: ο Kafka broker στον οποίο αποστέλεται το stream που δημιουργείται

topic: το topic στο οποίο ανήκει το stream

source: το path του αρχείου το οποίο θα χρησιμοποιηθεί ως πηγή για τη δημιουργία του stream

conf_path: το path του directory στο οποίο αναμένεται να βρεθούν όλα τα απαιτούμενα αρχεία configuration

timeout: ο χρόνος, σε δευτερόλεπτα, μέσα στον οποίο αποστέλεται η κάθε παρτίδα δεδομένων

Αφού κληθεί η produce_stream, ανοίγει το αρχείο source προς ανάγνωση και με βάση τα μεγέθη που θα βρει στο conf_path/batches αρχείο αρχίζει ανά timeout δευτερόλεπτα να παράγει κομμάτια ροής δεδομένων. Αφού ολοκληρωθεί η ανάγνωση και αποστολή μιας παρτίδας, η διεργασία κοιμάται μέχρι να συμπληρωθεί ο χρόνος του timeout και συνεχίζει μέχρι είτε να τελειώσουν τα δεδομένα, είτε να τελειώσουν τα μεγέθη παρτίδων που παρέχονται στο αρχείο conf_path/batches.

## *3.2 Εξαγωγή στατιστικών*

Προκειμένου να ενεργοποιηθούν οι βελτιστοποιήσεις κόστους του Catalyst, είναι απαραίτητη προϋπόθεση η ύπαρξη στατιστικών. Όπως αναφέρθηκε και στον τρόπο λειτουργίας των βελτιστοποιήσεων κόστους, τα ελάχιστα απαιτούμενα στατιστικά είναι το μέγεθος του DataFrame σε bytes και το πλήθος των εγγραφών. Για περισσότερη ακρίβεια στις εκτιμήσεις ενδιάμεσων μεγεθών χρησιμοποιούνται και μεγέθη όπως η μέγιστη και ελάχιστη τιμή, το πλήθος των διακριτών τιμών κάθε στήλης, καθώς και το μέγιστο και μέσο μήκος εγγραφών σε κάθε στήλη.

Η συλλογή στατιστικών για τα στατικά δεδομένα είναι μια απλή διαδικασία που έγινε μια φορά στην αρχή, καθώς τα στατικά δεδομένα παραμένουν σταθερά. Σε περίπτωση που τα δεδομένα βρίσκονται σε Hive format, αρκεί η εντολή “ANALYZE TABLE table COMPUTE STATISTICS”. Στη δική μας περίπτωση, όπου τα στατικά δεδομένα βρίσκονται σε αρχεία parquet, χρησιμοποιήθηκαν συναρτήσεις του DataFrame API και aggregations, όπως το describe, count, avg, max, min.

Για τις ροές δεδομένων, η εξαγωγή των στατιστικών γίνεται ανά παρτίδα, παράλληλα με την παραγωγή της ροής. Κατά την αποστολή της κάθε εγγραφής προς τον Kafka server, γίνεται ανάλυσή της και κρατάμε τα απαιτούμενα στατιστικά.

Καθώς ένας από τους στόχους της διπλωματικής είναι και ο προσδιορισμός της επιρροής των στατιστικών στην ποιότητα της βελτιστοποίησης, ανάλογα με το είδος και την ακρίβειά τους, η εξαγωγή των στατιστικών από τις ροές δεδομένων παραμετροποιείται με βάση τις παρακάτω παραμέτρους:

- detailed
- stats_accuracy
- stats_estimation

Ανεξαρτήτως παραμέτρων, υπολογίζουμε πάντα το μέγεθος της παρτίδας σε Bytes και το πλήθος των εγγραφών που αποστέλουμε. Η υπολογισμός γίνεται εύκολα και γρήγορα με χρήση counter.

Η παράμετρος detailed, ανάλογα με το αν είναι αληθής ή όχι, ενεργοποιεί την εξαγωγή αναλυτικών στατιστικών ανά στήλη για κάθε εγγραφή. Όταν είναι αληθής, τα εξαγώμενα στατιστικά, πέρα από το μέγεθος σε Bytes και το πλήθος εγγραφών, περιέχουν το πλήθος των διακριτών τιμών ανά στήλη και τις μέγιστες και ελάχιστες τιμές κάθε στήλης. Οι μέγιστες και ελάχιστες τιμές κατασκευάζονται καθώς διαβάζουμε την κάθε εγγραφή, αποφεύγοντας έτσι την ανάγκη για full scan της παρτίδας εκ των υστέρων. Το πλήθος των διακριτών τιμών υπολογίζεται προσθέτοντας την κάθε τιμή σε ένα set και υπολογίζοντας στο τέλος το μέγεθός του.

Η παράμετρος stats_accuracy παίρνει τιμές από 0 εως 1 και καθορίζει το ποσοστό του δείγματος απ το οποίο θα εξαχθούν τα στατιστικά. Όταν παίρνει την τιμή 1, τα στατιστικά εξάγονται απ το σύνολο των εγγραφών και είναι απόλυτα. Αν πάρει τιμή 0.1, τα στατιστικά προκύπτουν από δειγματοληψία 10% του συνόλου των εγγραφών.

Η παράμετρος stats_estimation, ανάλογα με το αν είναι αληθής ή όχι, ενεργοποιεί την παραγωγή στατιστικών με βάση ιστορικά στοιχεία αντί για την εξαγωγή τους. Οι τιμές του πλήθους των διακριτών τιμών για κάθε στήλη υπολογίζονται από ιστορικά ποσοστά επί του συνολικού πλήθους εγγραφών της παρτίδας. Με αυτόν τον τρόπο ο υπολογισμός γίνεται πολύ πιο γρήγορα, καθώς δεν έχουμε ανάλυση της κάθε εγγραφής, αλλά οδηγεί σε στατιστικά περιορισμένης ακρίβειας.

Οι παραπάνω παράμετροι και ο τρόπος εξαγωγής χρησιμοποιήθηκαν στην πειραματική αξιολόγηση της διπλωματικής. Μετά το πέρας των μετρήσεων, δοκιμάσαμε και τα παρακάτω ως επέκταση της εξαγωγής στατιστικών:

- Αποσύμπλεξη της εξαγωγής στατιστικών από τη δημιουργία της ροής δεδομένων. Η εξαγωγή στατιστικών κατά τη δημιουργία της ροής δεδομένων, παραπέμπει σε σενάριο όπου ο πάροχος των δεδομένων φροντίζει να παρέχει και στατιστικά επί αυτών. Σε μια απόπειρα προσέγγισης πιο ρεαλιστικού σεναρίου, η εξαγωγή στατιστικών αποσυμπλέχθηκε από τη δημιουργία ροής και μπήκε σε ξεχωριστή διεργασία, η οποία αναλύει την ροή, εξάγει στατιστικά και δημιουργεί νέα ροή δεδομένων με τις εγγραφές που έχει ήδη αναλύσει και για τις οποίες μπορεί να παρέχει άμεσα στατιστικά.
- Ενσωμάτωση στο spark streaming app. Μια άλλη απόπειρα αποσύμπλεξης της εξαγωγής στατιστικών από την δημιουργία των ροών δεδομένων ήταν ο υπολογισμός των στατιστικών εντός της streaming εφαρμογής. Τα πρώτα

αποτελέσματα ήταν πολύ αποθαρρυντικά από πλευράς καθυστέρησης και δε συνεχίστηκε η προσπάθεια.

Τα πλεονέκτημα της ταυτόχρονης εξαγωγής στατιστικών με τη δημιουργία της ροής και του stream pre-processing έναντι της εξαγωγής εντώς μια εφαρμογής spark streaming, είναι πως τα στατιστικά παράγονται μια φορά και είναι άμεσα διαθέσιμα σε όλες τις streaming εφαρμογές που μπορεί να τα ζητήσουν και επιπρόσθετα, ο χρόνος που καταναλώνεται για την εξαγωγή των στατιστικών γίνεται στο χρονικό περιθώριο που οι streaming εφαρμογές αναμένουν να γεμίσει ο buffer με δεδομένα, και όχι στο χρόνο εκτέλεσης της ίδιας της streaming εφαρμογής.

Μια τελευταία επέκταση που δοκιμάστηκε και φάνηκε πολλά υποσχόμενη, είναι η αποσύμπλεξη της εξαγωγής στατιστικών από την έννοια της παρτίδας. Με χρήση του αλγορίθμου Sliding HyperLogLog και εξαγωγή των υπόλοιπων στατιστικών ανά σταθερά χρονικά διαστήματα, έγινε δυνατό να παρέχουμε on demand στατιστικά για ό,τι χρονικό διάστημα ζητηθεί, με πολύ μικρές απαιτήσεις μνήμης.

## *3.3 Έγχυση στατιστικών*

Αφού εξάγουμε τα στατιστικά, προκειμένου να λειτουργήσουν οι βελτιστοποιήσεις κόστους του Catalyst, πρέπει με κάποιον τρόπο να του τα παρέχουμε όπου αυτός τα χρειάζεται. Η έκδοση του spark που χρησιμοποιήθηκε δεν παρέχει δυνατότητα παροχής έτοιμων στατιστικών.

Όπως αναλύθηκε και στον τρόπο λειτουργίας του Catalyst, ένα λογικό πλάνο αποτελείται από επιμέρους λογικά πλάνα. Τα στατιστικά ενός λογικού πλάνου υπολογίζονται με βάση τον τύπο του και τα στατιστικά των παιδιών-πλάνων. Κάθε operator υλοποιεί τη συνάρτηση computeStats, η οποία καθορίζει το πως προκύπτουν τα νέα στατιστικά. Προκειμένου να αποφευχθεί πολλαπλός υπολογισμός των στατιστικών ενός λογικού πλάνου, το Spark φροντίζει ώστε το κάθε λογικό πλάνο να διατηρεί StatsCache με τα υπολογισμένα στατιστικά, και σε περίπτωση που αυτή είναι κενή ή έχει μαρκαριστεί ως μη έγκυρη, επιχειρεί να τα υπολογίσει ξανά.

Προκειμένου να επιτύχουμε το στόχο έγχυσης στατιστικών στοχευμένα σε DataFrames της επιλογής μας, διατηρώντας την επέμβαση στον πυρήνα του Spark όσο το δυνατόν μικρότερη, επιλέχθηκε η έγχυση να γίνει μέσω επέμβασης της StatsCache. Μέσω reflection, επιτυγχάνεται πρόσβαση και επέμβαση στο κανονικά read only πεδίο StatsCache, το οποίο και θέτουμε στην τιμή που εμείς θέλουμε.

Στα DataFrames των στατικών δεδομένων που προκύπτουν από τα parquet αρχεία, αρκεί να γίνει έγχυση στο DataFrame που παράγεται από την ανάγνωση των αρχείων.

Στην περίπτωση των ροών, χρειάστηκε μια ακόμα επέμβαση προκειμένου να λειτουργήσει σωστά η έγχυση στατιστικών. Τα DataFrames που παράγονται από τα RDDs του Spark Streaming ανατίθενται σε πλάνα-γονείς τύπου SerializeFromObject. Η κλάση SerializeFromObject δεν υλοποιεί την ComputeStats και επομένως τρέχει η default, η οποία αναθέτει στα στατιστικά μια αυθαίρετη τιμή. Η επέμβαση που έγινε σε αυτή την περίπτωση, είναι η υλοποίηση της computeStats στην κλάση SerializeFromObject ώστε να ανιχνεύεται η ύπαρξη στατιστικών στο πλάνο παιδί τους και να τα διατηρούν όταν αυτά υπάρχουν.

## *3.4 Spark Streaming App*

Σε αυτό το στάδιο, έχουμε πλέον επιτύχει την εξαγωγή στατιστικών από τις ροές δεδομένων και υλοποιήσει τον μηχανισμό έγχυσής τους στα DataFrames όπου και θα τα βρει ο Catalyst για να τα αξιοποιήσει. Επόμενο βήμα είναι η εφαρμογή Spark Streaming, η οποία και θα επιτύχει την προσαρμοστική βελτιστοποίηση του πλάνου εκτέλεσης του ερωτήματος, καθώς θα εγχύει διαρκώς τα ανανεωμένα στατιστικά των ροών, επιτρέποντας έτσι στον Catalyst να καταστρώνει κάθε φορά πλάνο που να ταιριάζει στην νέα κατάσταση.

Η εφαρμογή Spark Streaming είναι γραμμένη σε Scala. Ο πυρήνας της εφαρμογής είναι η κλάση AdaptiveQuery

```
case class AdaptiveQuery(
sc: SparkContext, spark: SparkSession, hdfsPath: String = "tpch/tpch-
s50", confDir: String, interval: Int, window, statsAccuracy: String,
querySource: String = "dataframe") {
```

hdfsPath: το path στο οποίο βρίσκονται τα στατικά δεδομένα στο hdfs

confDir: το path στο οποίο βρίσκεται το αρχείο με τα στατιστικά των στατικών δεδομένων

interval: χρονικό διάστημα σε δευτερόλεπτα στην ολοκλήρωση του οποίου κάθε φορά τρέχει η Spark Streaming εφαρμογή μας

window: Μήκος παραθύρου δεδομένων που χρησιμοποιούνται στην ανάλυση που τρέχει κάθε φορά η Spark Streaming εφαρμογή μας

statsAccuracy: detailed/basic/none. Ακρίβεια στατιστικών προς έγχυση στα DataFrames

querySource: Επιλογή μορφής του query που θα χρησιμοποιηθεί, μεταξύ DataFrame/SQL

Με τη δημιουργία ενός AdaptiveQuery object, ετοιμάζονται τα DataFrames με τα στατικά δεδομένα από τα parquet αρχεία που βρίσκονται στο hdfs, γίνεται έγχυση στατιστικών και προετοιμάζεται η ροή δεδομένων από Kafka, δημιουργώντας και εγγράφοντας Kafka καταναλωτές στα κατάλληλα topics.

Στη συνέχεια, μέσω της execute συνάρτησης της κλάσης, δίνεται το ερώτημα προς εκτέλεση, περιγράφεται η διαδικασία που εκτελείται ανά κάθε παρτίδα και εκκινείται το Spark

Streaming Context. Η διαδικασία ανά παρτίδα είναι η εξής: Αρχικά αποσυμπλέκονται τα topics του Kafka Stream σε διαφορετικά RDDs, τα RDDs μετατρέπονται σε DataFrames, στα DataFrames αυτά γίνεται η έγχυση των τελευταίων στατιστικών της κάθε ροής δεδομένων, εφαρμόζεται το query στα DataFrames, στατικά και ροών, τα οποία έχουν πλέον τα στατιστικά, και παράγεται και εκτελείται το βελτιστοποιημένο πλάνο.

Κατά τη διάρκεια της εκτέλεσης, μέσω της διαδικτυακής διεπαφής του Spark, παρακολουθούμε την εξέλιξη των ερωτημάτων, τα τελικά πλάνα που εκτελούνται και τους συνολικούς χρόνους εκτέλεσής τους.

# 4

# *Πειραματική Αξιολόγηση*

Προκειμένου να αξιολογήσουμε την αποτελεσματικότητα της προσαρμοστικής βελτιστοποίησης εκτέλεσης ερωτημάτων θα τρέξουμε Χ ερωτήματα εναλλάσσοντας τους όγκους των ροών δεδομένων καθώς και την ακρίβεια των στατιστικών που εξάγουμε από αυτές.

## *4.1 Πειραματική Διάταξη*

Για την εκτέλεση των πειραμάτων δημιουργήσαμε ένα cluster από υπολογιστές στις υποδομές OpemStack του cslab. Η διάταξη αποτελείται από τα εξής VMs

**Table 4-1 Προδιαγραφές Cluster**

| | CPUs | RAM |
|---|---|---|
| Master | 8x2.4GHz Intel Core i7 | 16GB |
| Slave (x3) | 4x2.4GHz Intel Core i7 | 8GB |
| Stream Server | 4x2.4GHz Intel Core i7 | 16GB |

Οι διεργασίες που τρέχουν στον Master είναι οι εξής: JobTracker, NameNode και DataNode του Hadoop, master, workers (x4), driver, history server του Spark.

Οι διεργασίες που τρέχουν στους Slaves είναι οι εξής: DataNode του Hadoop, workers (x4) του Spark.

Στον Stream Server τρέχει Zookeeper server, Kafka server, Kafka stream producers και Redis server.

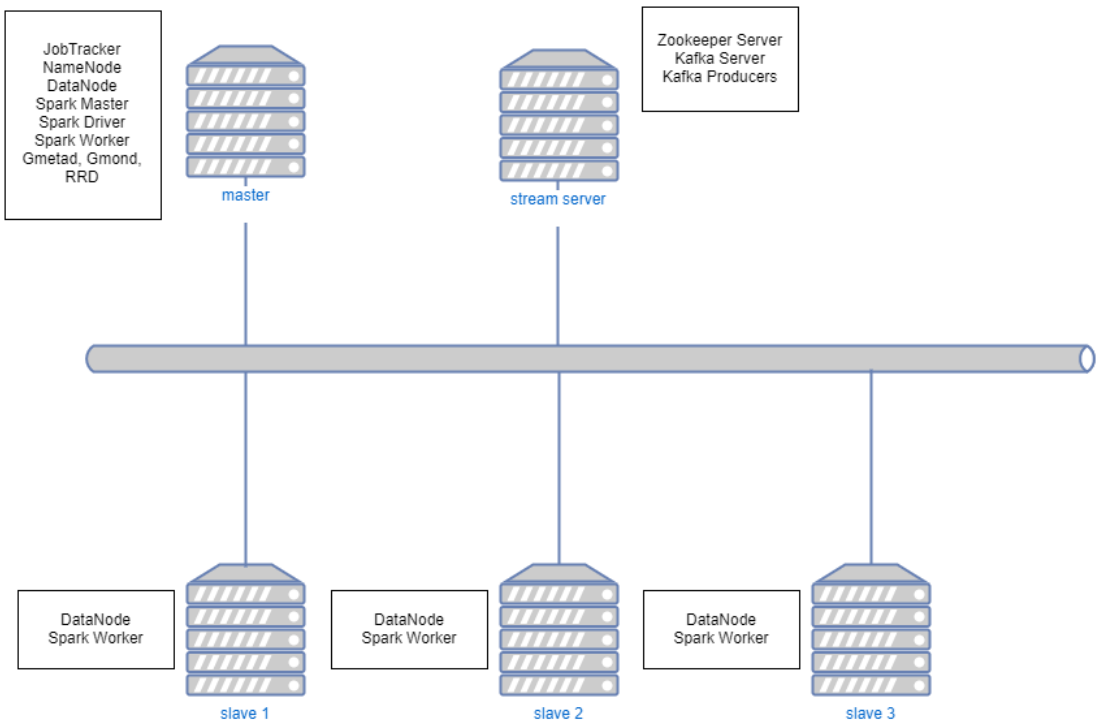


**Figure 4-1 Αρχιτεκτονική πειραματικής διάταξης**

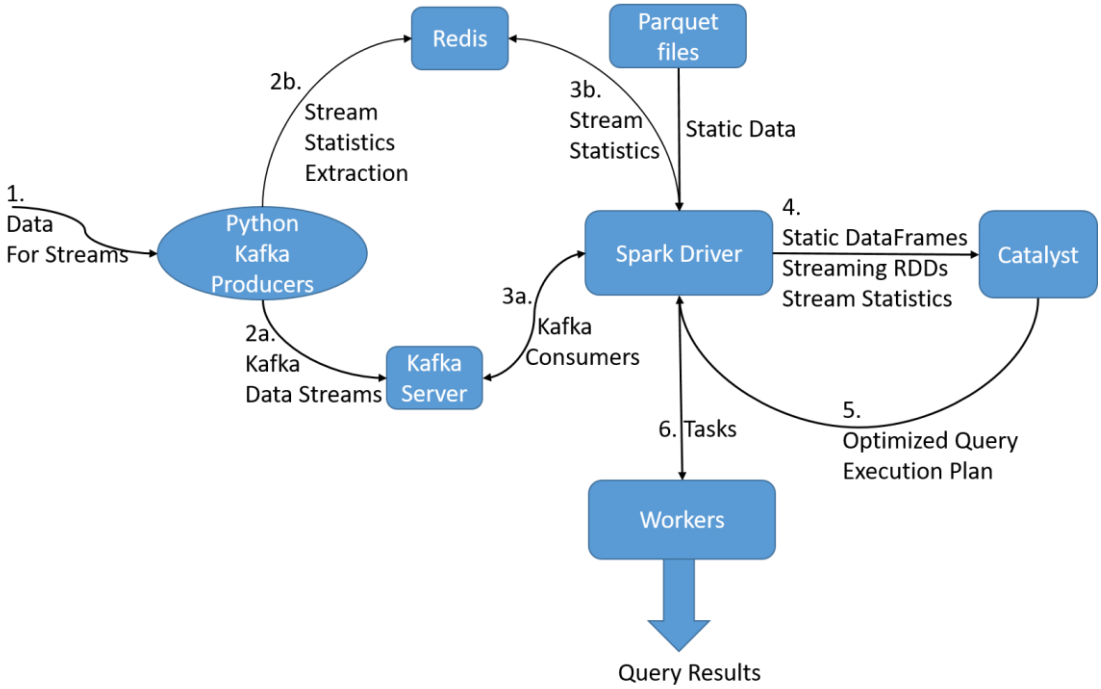


**Figure 4-2 Ροή Προσαρμοστικής Βελτιστοποίησης Εκτέλεσης Ερωτημάτων**

Το HDFS χρησιμοποιείται ως DataStore του Spark. Εκεί βρίσκονται αποθηκευμένα τα στατικά δεδομένα μας σε μορφή parquet αρχείων καθώς και τα logs του history server.

Ο Redis Server χρησιμοποιείται ως εύκολος τρόπος πρόσβασης στα στατιστικά που παράγει ο Stream Server από τον Spark Driver που τα χρησιμοποιεί.

Ο Kafka Server χρησιμοποιείται ως η πηγή των ροών δεδομένων μας. Οι python Kafka Producers φροντίζουν να παράγουν τις ροές δεδομένων, με διαρκώς μεταβαλλόμενο ρυθμό ο οποίος καθορίζεται από παραμέτρους εκτέλεσής τους, καθώς και να εξάγουν στατιστικά από αυτές και να ανανεώνουν διαρκώς την Redis cache.

Ο Spark Driver, που τρέχει στον Master κόμβο, τρέχει την εφαρμογή Spark Streaming, συλλέγει και προετοιμάζει με έγχυση στατιστικών τα στατικά δεδομένα, εγγράφεται στις ροές δεδομένων Kafka, και ανά σταθερά χρονικά διαστήματα που ορίζουμε, δημιουργεί RDDs από τις ροές δεδομένων, εγχύει τα τελευταία στατιστικά που λαμβάνει από τον Redis Server, βελτιστοποιεί με τη βοήθεια του Catalyst το πλάνο προς εκτέλεση, δημιουργεί τα απαιτούμενα tasks και τα κατανέμει στους Workers που βρίσκονται στον κόμβο Master και τους 3 Slaves.

## *4.2 Επιλογή Dataset και ερωτημάτων*

Ως dataset για την εκτέλεση των πειραμάτων επιλέχθηκε το TPC-H: ένα benchmark λήψης αποφάσεων. Επιλέχθηκε λόγω των απαιτητικών από πλευράς συνενώσεων ερωτημάτων που περιλαμβάνει, καθώς ερωτήματα τέτοιου τύπου πιστεύουμε πως θα ωφεληθούν περισσότερο από την προσαρμοστική βελτιστοποίησή τους.

Το TPC-H αποτελείται από 8 πίνακες, όπως αυτοί φαίνονται στο παρακάτω διάγραμμα.

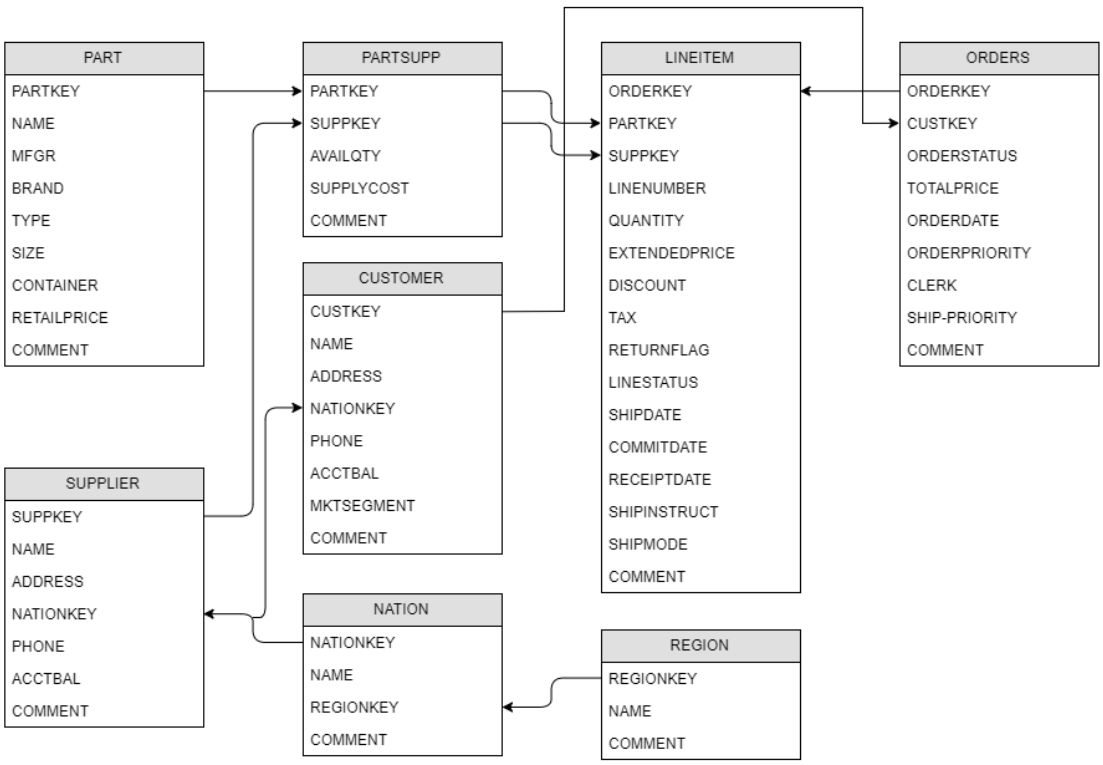


**Figure 4-3 TPCH schema**

Στα πλαίσια των πειραμμάτων μας, οι πίνακες ORDERS και LINEITEM μετατράπηκαν σε ροές δεδομένων, ενώ οι υπόλοιποι πίνακες χρησιμοποιήθηκαν ως στατικά δεδομένα.

Το TPC-H περιλαμβάνει 21 ερωτήματα για τα δεδομένα του. Από αυτά, επιλέχθηκαν προς διερεύνηση τα 3 πιο απαιτητικά πλάνα από πλευράς ενώσεων που ταυτόχρονα περιέχουν και τις δυο ροές δεδομένων μας.

Τα ερωτήματα που επιλέχθηκαν είναι τα Q8, Q9 και Q21.

Q8

```
select
  o_year,
  sum(case
    when nation = 'BRAZIL' then volume
    else 0
  end) / sum(volume) as mkt_share
from
  (
    select
      year(o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) as volume,
      n2.n_name as nation
    from
      part,
      supplier,
      lineitem,
```

```
      orders,
      customer,
      nation n1,
      nation n2,
      region
    where
      p_partkey = l_partkey
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'AMERICA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between date '1995-01-01' and date '1996-12-
31'
      and p_type = 'ECONOMY ANODIZED STEEL'
  ) as all_nations
group by
  o_year
order by
  o_year
```

Q9

```
select
  nation,
  o_year,
  sum(amount) as sum_profit
from
  (
    select
      n_name as nation,
      year(o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) - ps_supplycost *
l_quantity as amount
    from
      part,
      supplier,
      lineitem,
      partsupp,
      orders,
      nation
    where
      s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
```

```
  and p_name like '%green%'
  ) as profit
group by
  nation,
  o_year
order by
  nation,
  o_year desc
```

Q21

```
select
  s_name,
  count(*) as numwait
from
  supplier,
  lineitem l1,
  orders,
  nation
where
  s_suppkey = l1.l_suppkey
  and o_orderkey = l1.l_orderkey
  and o_orderstatus = 'F'
  and l1.l_receiptdate > l1.l_commitdate
  and exists (
    select
      *
    from
      lineitem l2
    where
      l2.l_orderkey = l1.l_orderkey
      and l2.l_suppkey <> l1.l_suppkey
  )
  and not exists (
    select
      *
    from
      lineitem l3
    where
      l3.l_orderkey = l1.l_orderkey
      and l3.l_suppkey <> l1.l_suppkey
      and l3.l_receiptdate > l3.l_commitdate
  )
  and s_nationkey = n_nationkey
  and n_name = 'SAUDI ARABIA'
group by
  s_name
order by
```

```
  numwait desc,
  s_name
limit 100
```

## 4.3 Κριτήρια Αξιολόγησης

Όπως προαναφέρθηκε, το Spark Streaming επεξεργάζεται τις ροές δεδομένων σε micro-batches, ανά σταθερά intervals. Επομένως, ως κριτήριο αξιολόγησης της απόδοσης της βελτιστοποίησης θα χρησιμοποιήσουμε τον συνολικό χρόνο επεξεργασίας για μια δεδομένη ροή δεδομένων. Καθορίζοντας από τους Kafka producers το μέγεθος των διαδοχικών micro-batches, θα εξετάσουμε σε ποια περίπτωση επιτυγχάνεται ο καλύτερος συνολικός χρόνος εκτέλεσης.

Θεωρούμε πως αυτό το κριτήριο αξιολόγησης αποτελεί έγκυρο κριτήριο σύγκρισης των διαφορετικών εκτελέσεων, καθώς ο μικρότερος χρόνος εκτέλεσης με την κατάλληλη παραμετροποίηση του spark, μπορεί να οδηγήσει σε αποδέσμευση spark executors και κατ επέκτασιν σε καλύτερη αξιοποίηση των υπολογιστικών πόρων.

Όλα τα πειράματα θα εκτελεστούν με τους ίδιους πόρους και θα συγκριθούν οι συνολικοί χρόνοι εκτέλεσης ανά παρτίδα.

## 4.4 Σενάρια και Αποτελέσματα

Για κάθε διαφορετικό ερώτημα εκτελούμε τα εξής σενάρια:

- Σταθερό πλάνο εκτέλεσης που παράγει το Spark Streaming μόνο του
  (Spark Streaming στα γραφήματα)
- Μεταβαλλόμενο πλάνο εκτέλεσης που παράγει ο Catalyst με χρήση μόνο των σειρών και του μεγέθους των ροών δεδομένων
  (Basic στα γραφήματα)
- Μεταβαλλόμενο πλάνο εκτέλεσης που παράγει ο Catalyst με χρήση εκτιμόμενων στατιστικών των ροών δεδομένων
  (Estimation στα γραφήματα)
- Μεταβαλλόμενο πλάνο εκτέλεσης που παράγει ο Catalyst με χρήση στατιστικών που προκύπτουν με δειγματολειψία των ροών δεδομένων
  (Sampling 1‰, 1% και 10% στα γραφήματα)
- Μεταβαλλόμενο πλάνο εκτέλεσης που παράγει ο Catalyst με χρήση απόλυτων στατιστικών που εξάγωνται από τις ροές δεδομένων
  (Absolute στα γραφήματα)

Για κάθε σενάριο αρχικά αναφέρονται τα batches ανάλυσης που έτρεξαν από το spark streaming. Επιλέξαμε μεγέθη batches τέτοια ώστε να παρατηρηθούν πολλαπλά διαφορετικά πλάνα ανάλογα με τη διακύμανση των δύο ροών. Ένας περιορισμός στην επιλογή των μεγεθών των batches είναι πως η βιβλιοθήκη που χρησιμοποιήθηκε από τους παραγωγούς ροών έχει άνω όριο buffer τα 10.000.000 μηνύματα, συνεπώς όλα τα batches τηρούν αυτό τον περιορισμό. Καθώς τα πειράματα εκτελέστηκαν σε ένα κατανεμημένο σύστημα, φροντίσαμε τα μεγέθη των batches να οδηγούν σε χρόνους εκτέλεσης αρκετά μεγάλους, ώστε οι διαφορές που εντοπίζονται να οφείλονται στα διαφορετικά πλάνα εκτέλεσης και όχι σε εξωτερικές παρεμβολές όπως το συνολικό φόρτο του δικτύου ή της υποδομής OpenStack. Προκειμένου να εξάγουμε όσο το δυνατόν πιο έγκυρες μετρήσεις, το κάθε πείραμα εκτελέστηκε πολλαπλές φορές και οι χρόνοι αποτελεσμάτων που παρατίθενται είναι αποτέλεσμα του μέσου όρου των πολλαπλών εκτελέσεων.

Στη συνέχεια, παρουσιάζονται 2 γραφήματα: το πρώτο παρουσιάζει τους απόλυτους χρόνους εκτέλεσης ανά batch και ανά μέθοδο παραγωγής πλάνου και το δεύτερο παρουσιάζει το speedup στο οποίο οδήγησε η κάθε μέθοδος παραγωγής πλάνου ανά batch.

Τέλος, παρατίθενται σχεδιαγράμματα των εξής πλάνων:

- Το πλάνο που προκύπτει από την ανάλυση του SQL ερωτήματος
- Το πλάνο που προκύπτει μετά τις λογικές βελτιστοποιήσεις του Catalyst
- Επιλογή πλάνων που προκύπτουν μετά τις φυσικές βελτιστοποιήσεις του Catalyst και παρουσιάζουν ενδιαφέρον λόγω των χρόνων εκτέλεσής τους

Αυτό που μπορούμε να παρατηρήσουμε στα διαφορετικά πλάνα είναι η εναλλαγή της σειράς εκτέλεσης των συνενώσεων, καθώς και η επιλογή του αλγορίθμου συνένωσης.

Προκειμένου να ξεχωρίζει ο αλγόριθμος συνένωσης, τα JOIN με συνεχή γραμμή είναι Sort Merge Join και τα JOIN που απεικονίζονται με διακεκομμένη γραμμή είναι Broadcast Hash Join.

## Q 8

Batch sizes

| | Batch 1 | Batch 2 | Batch 3 | Batch 4 | Batch 5 | Batch 6 | Batch 7 |
|---|---|---|---|---|---|---|---|
| Lineitem | 1K | 9M | 200K | 7M | 8M | 5M | 9M |
| Orders | 1K | 8M | 7M | 1M | 600K | 5M | 9M |

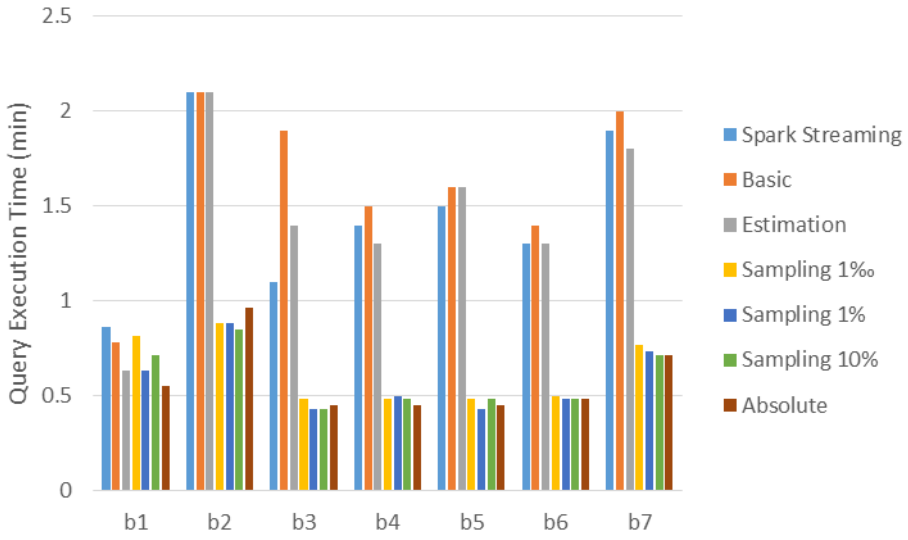


**Figure 4-4 Q8 Query Execution Time**

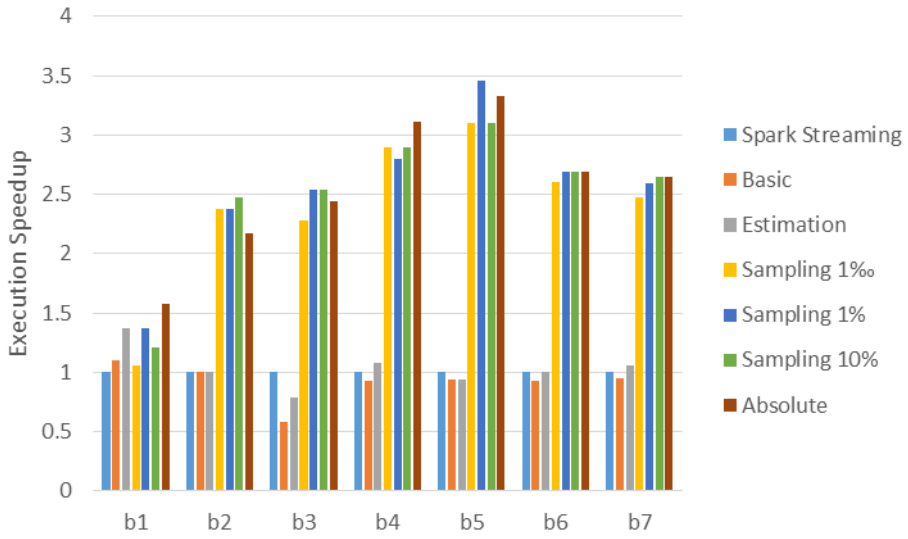


**Figure 4-5 Q8 Query Execution Speedup**

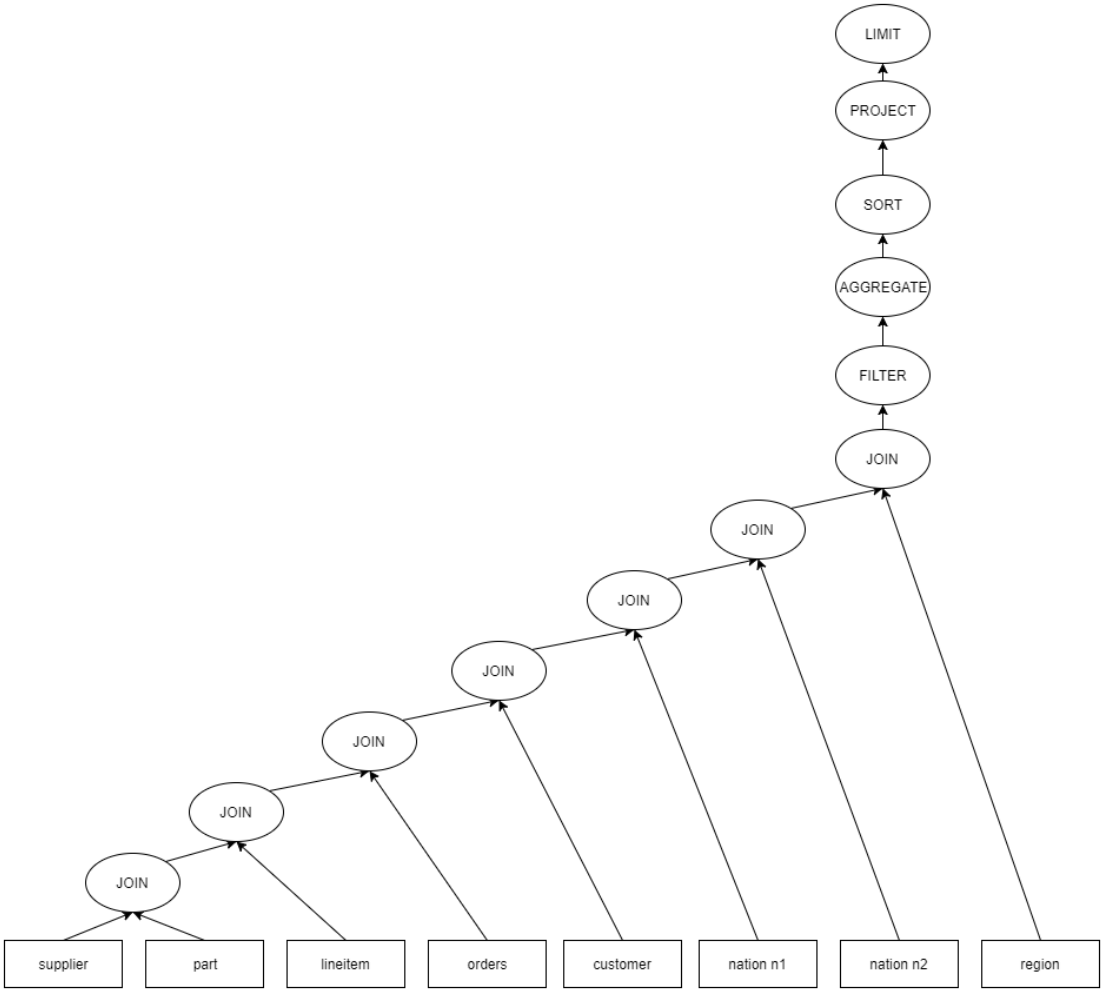


**Figure 4-6 Q8 Analyzed Plan**

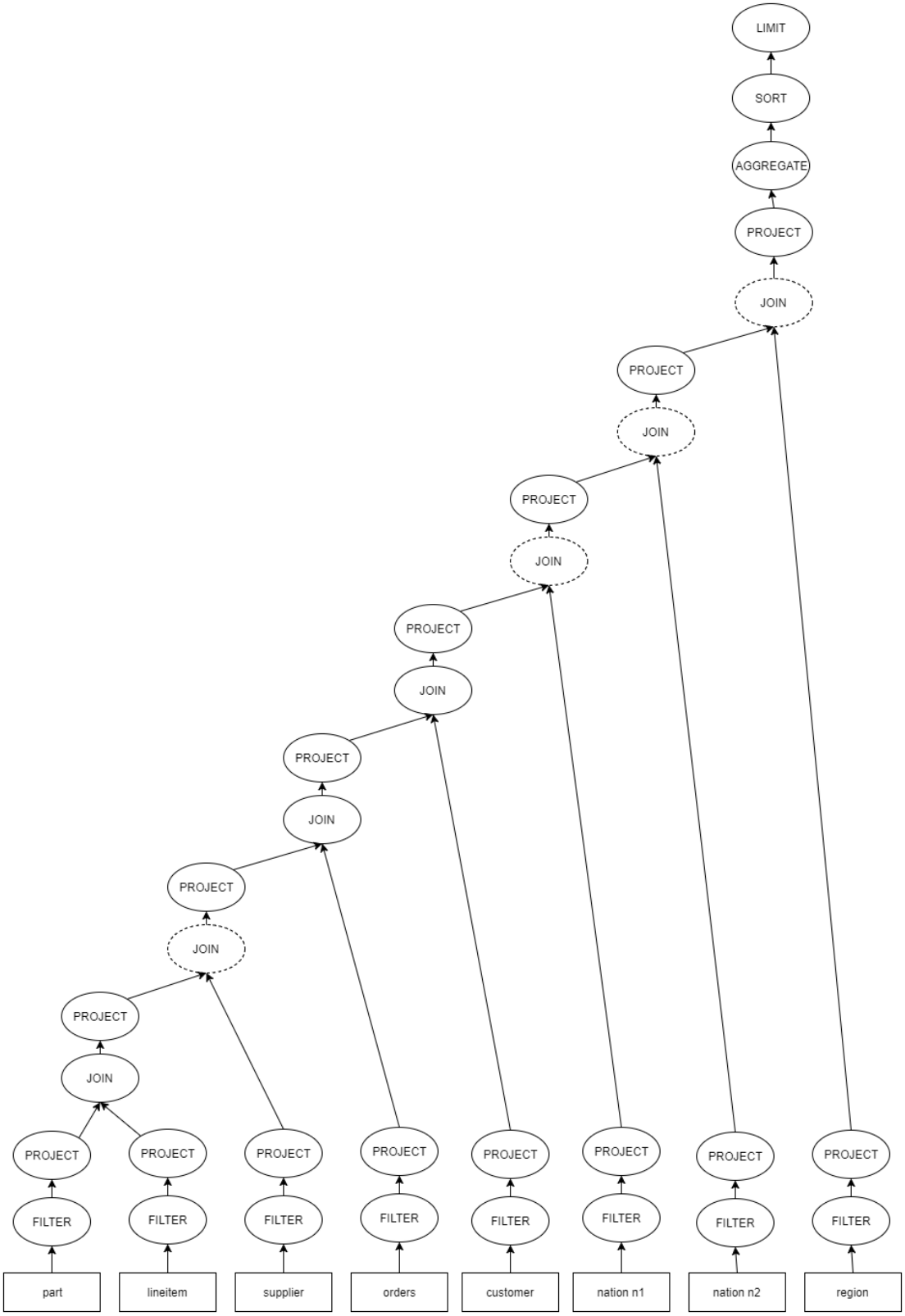


**Figure 4-7 Q8 Logical Optimizations Plan**

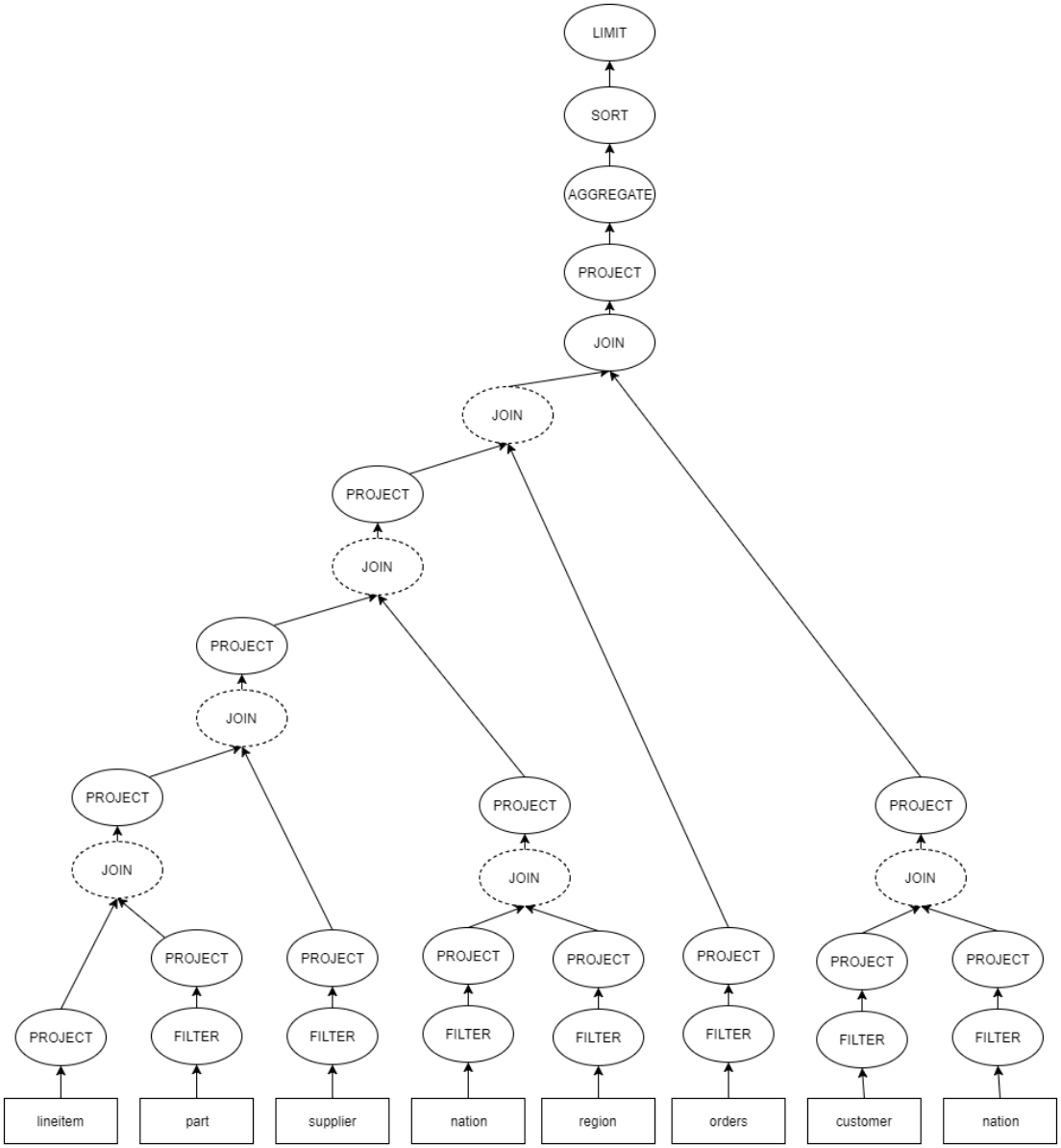


**Figure 4-8 Q8 Cost Based Optimizations Plan Batch 3**

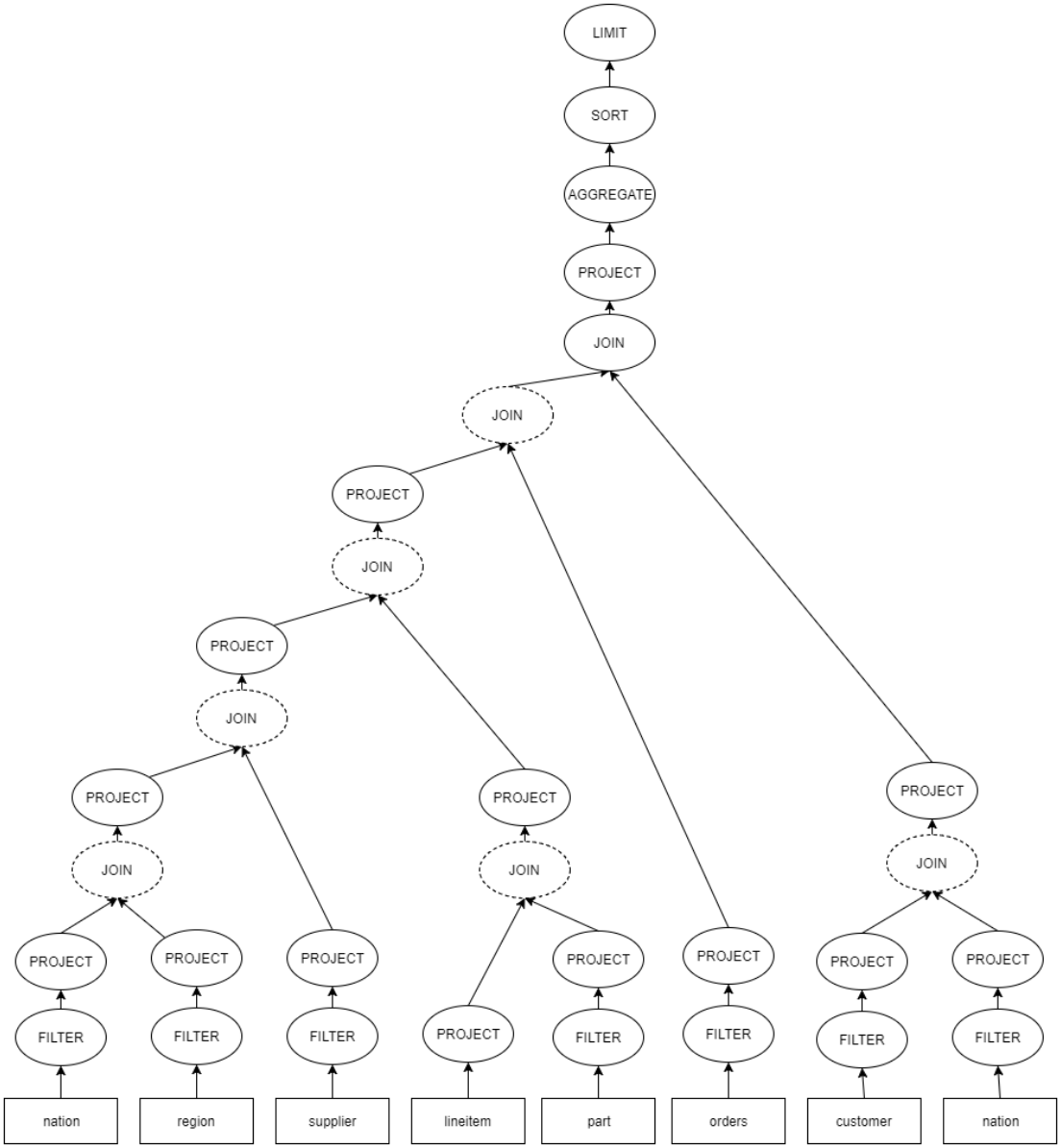


**Figure 4-9 Q8 Cost Based Optimizations Plan Batch 2**

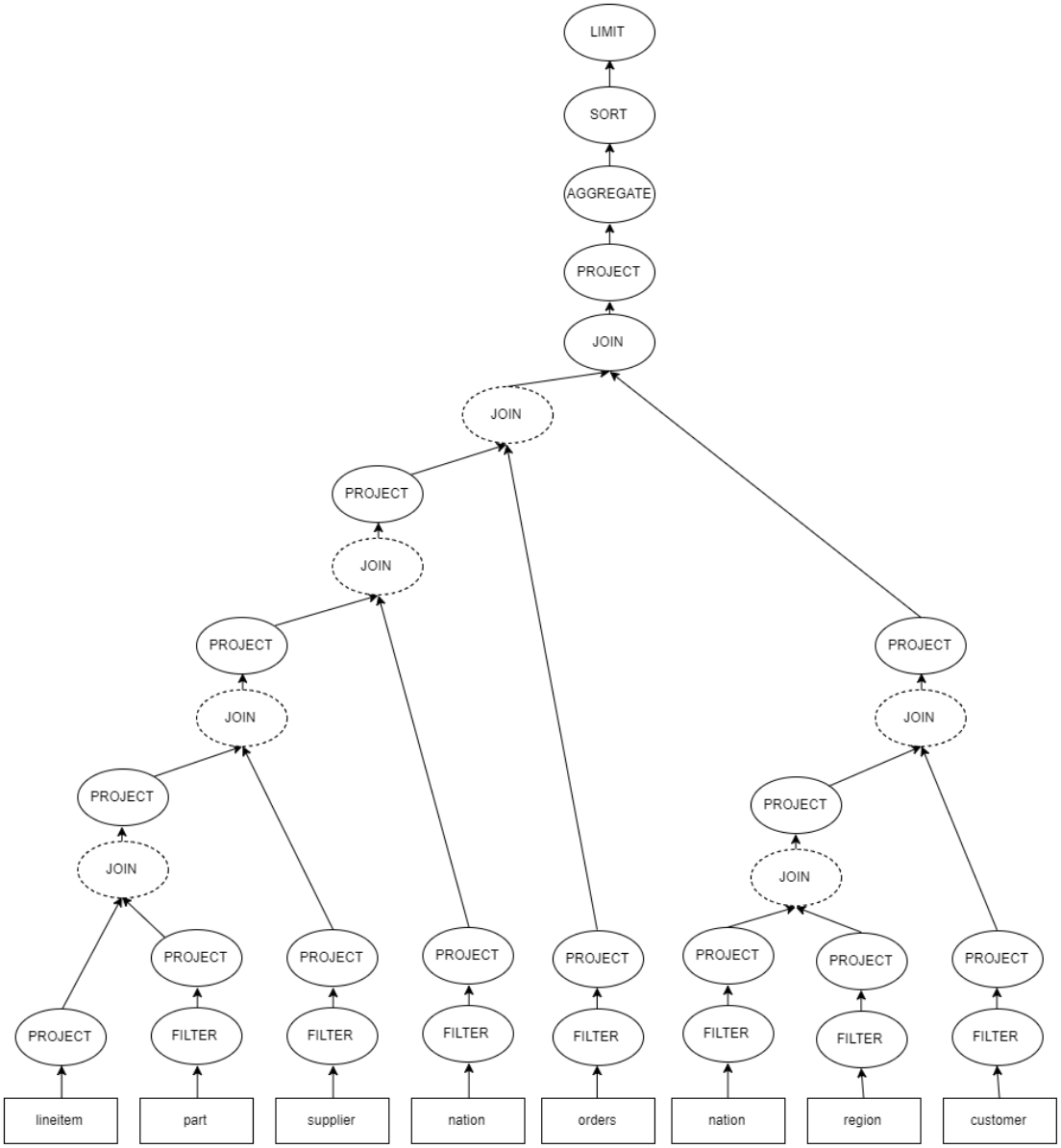


**Figure 4-10 Q8 Cost Based Optimizations Plan Batch 1**

## Q 9

Batch sizes

| | Batch 1 | Batch 2 | Batch 3 | Batch 4 | Batch 5 | Batch 6 | Batch 7 |
|---|---|---|---|---|---|---|---|
| Lineitem | 1K | 9M | 200K | 7M | 8M | 5M | 9M |
| Orders | 1K | 8M | 7M | 1M | 600K | 5M | 9M |

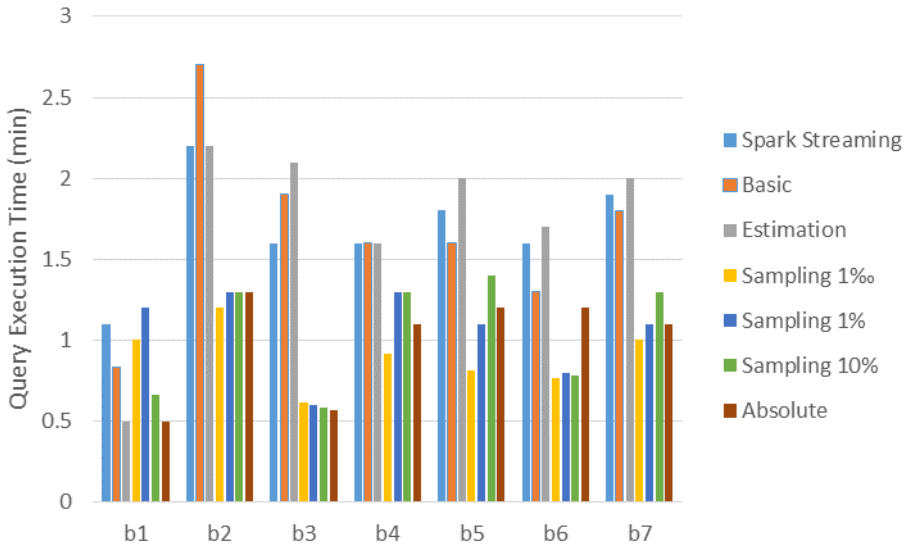


**Figure 4-11 Q9 Query Execution Time**

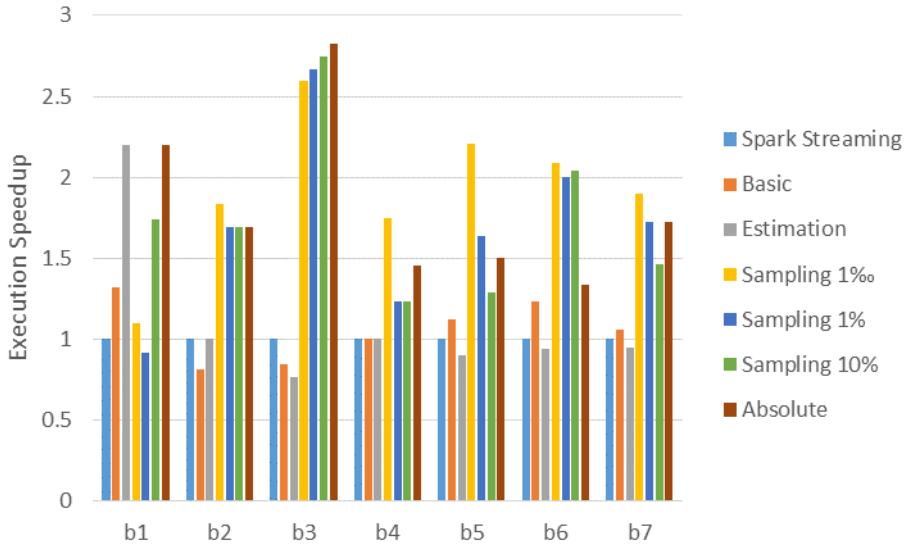


**Figure 4-12 Q9 Query Execution Speedup**

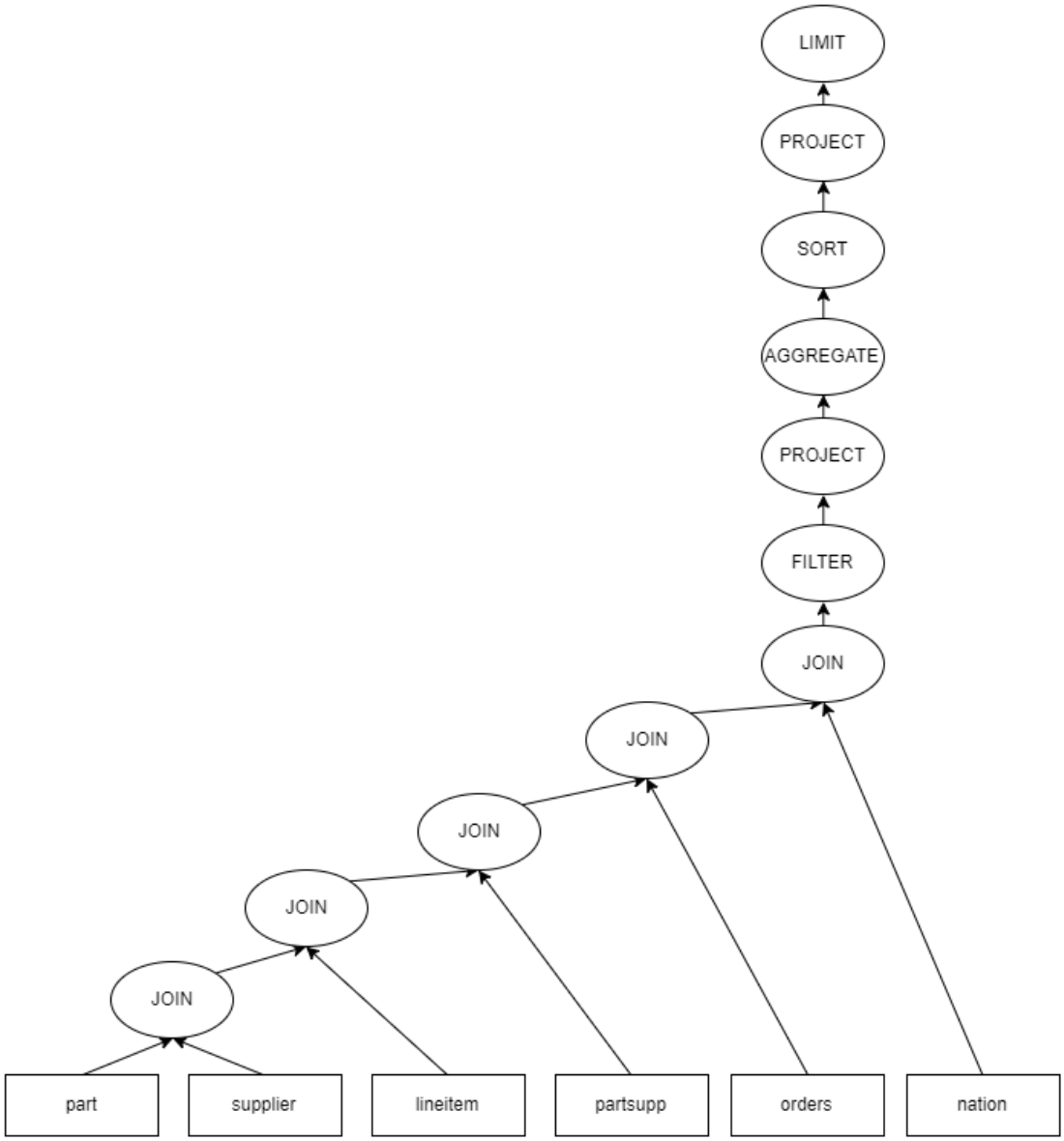


**Figure 4-13 Q9 Analyzed Plan**

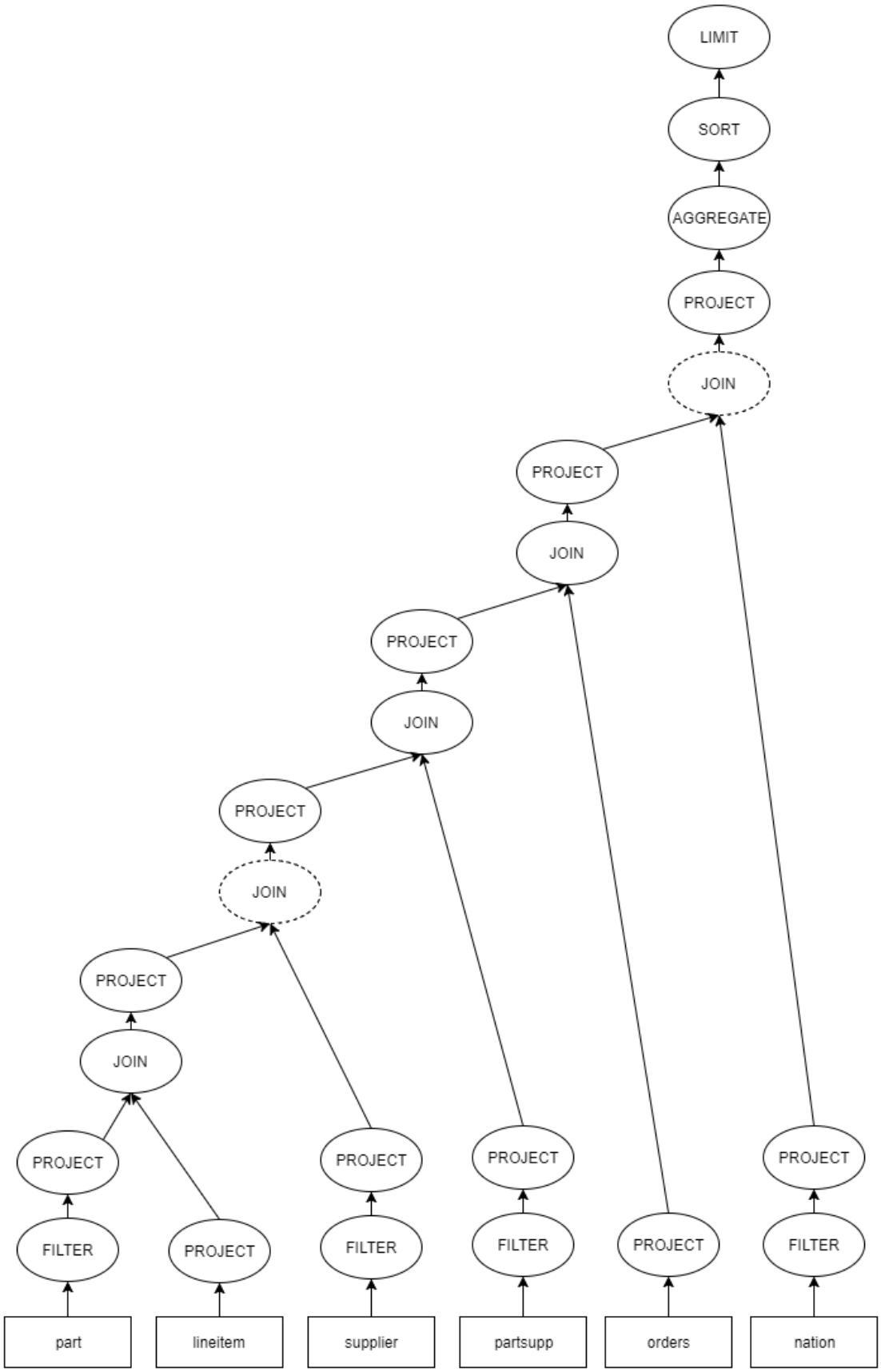


**Figure 4-14 Q9 Logical Optimizations Plan**

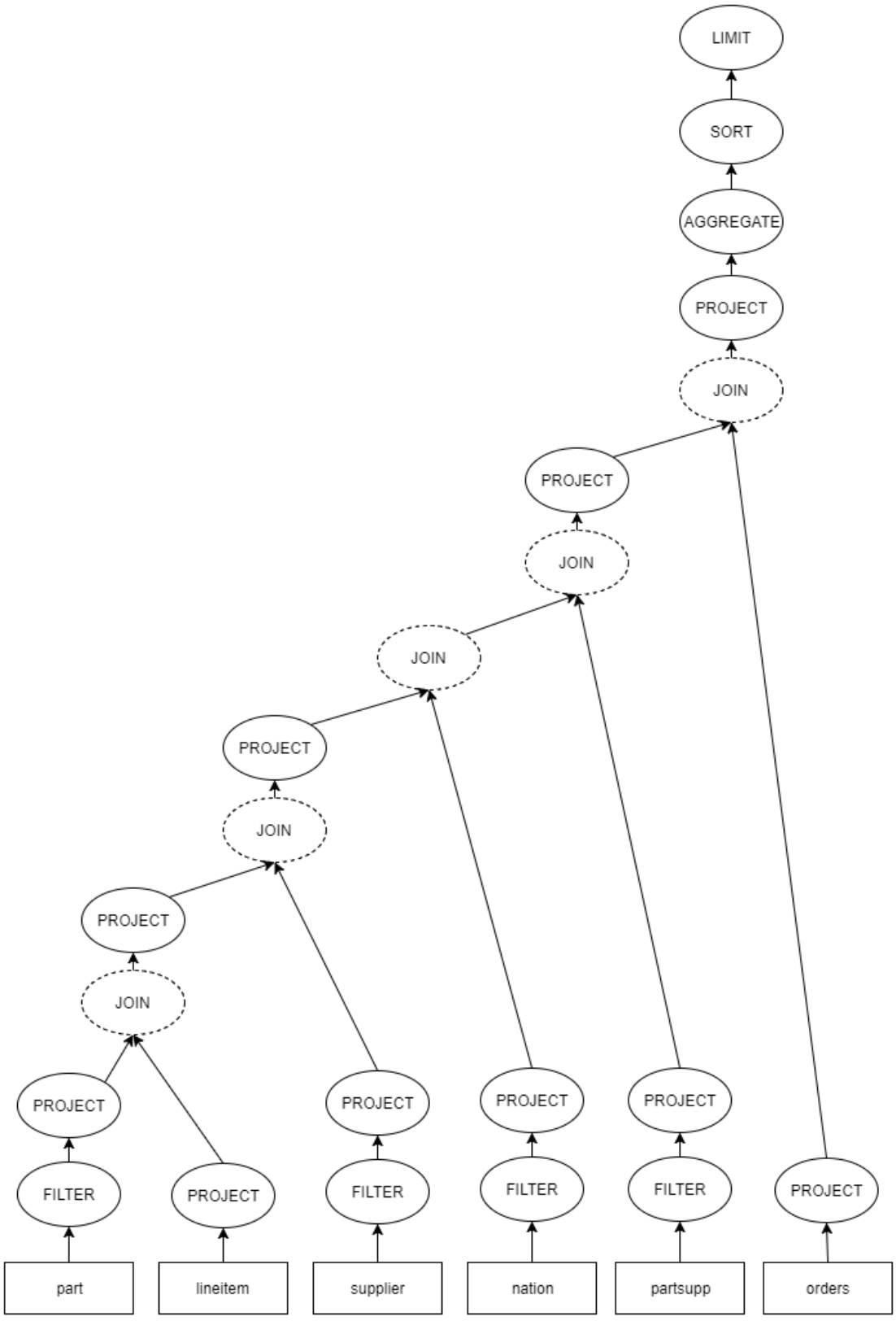


**Figure 4-15 Q9 Cost Based Optimizations Plan Batch 1**

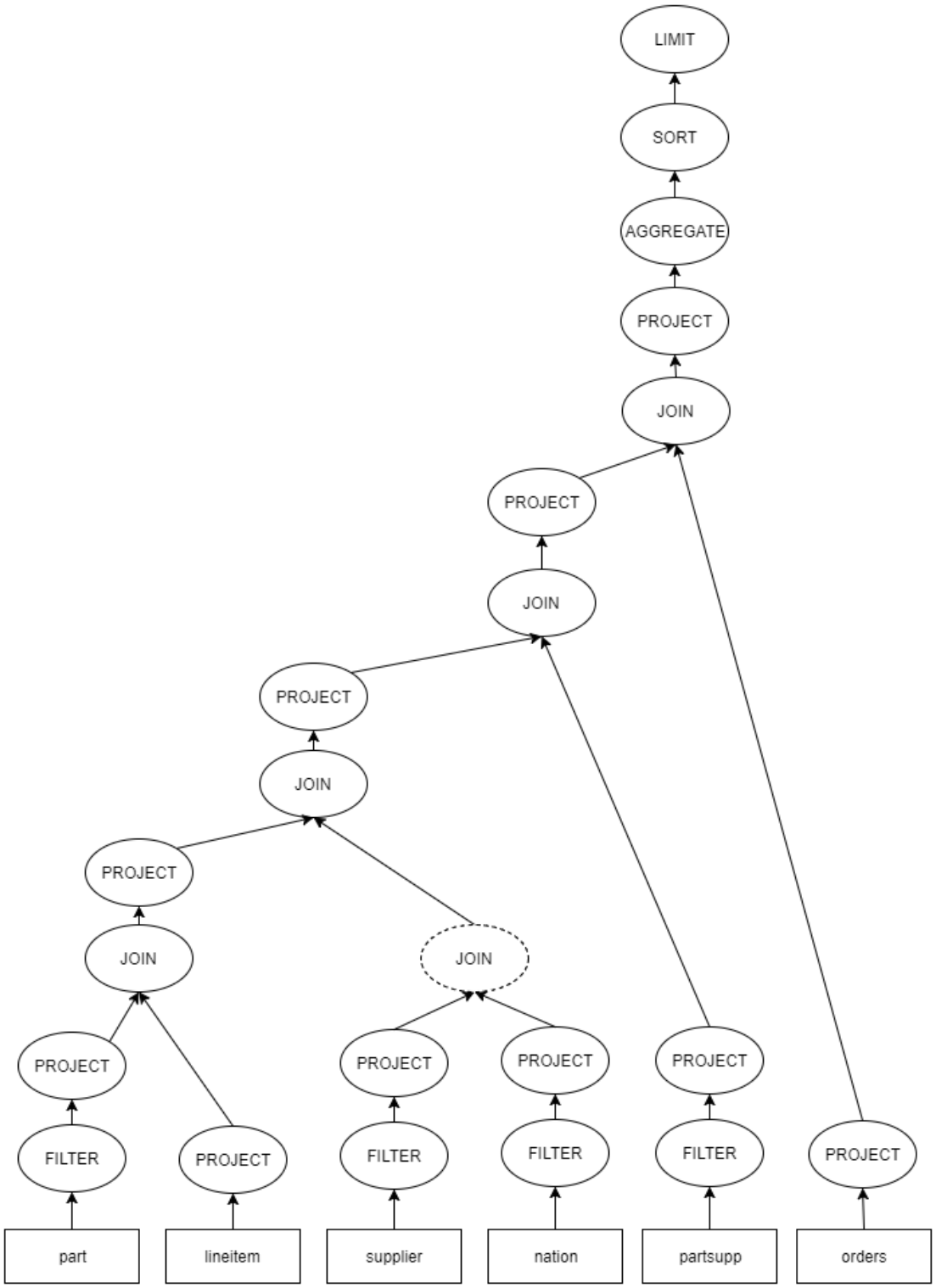


**Figure 4-16 Q9 Cost Based Optimizations Plan Batch 2**

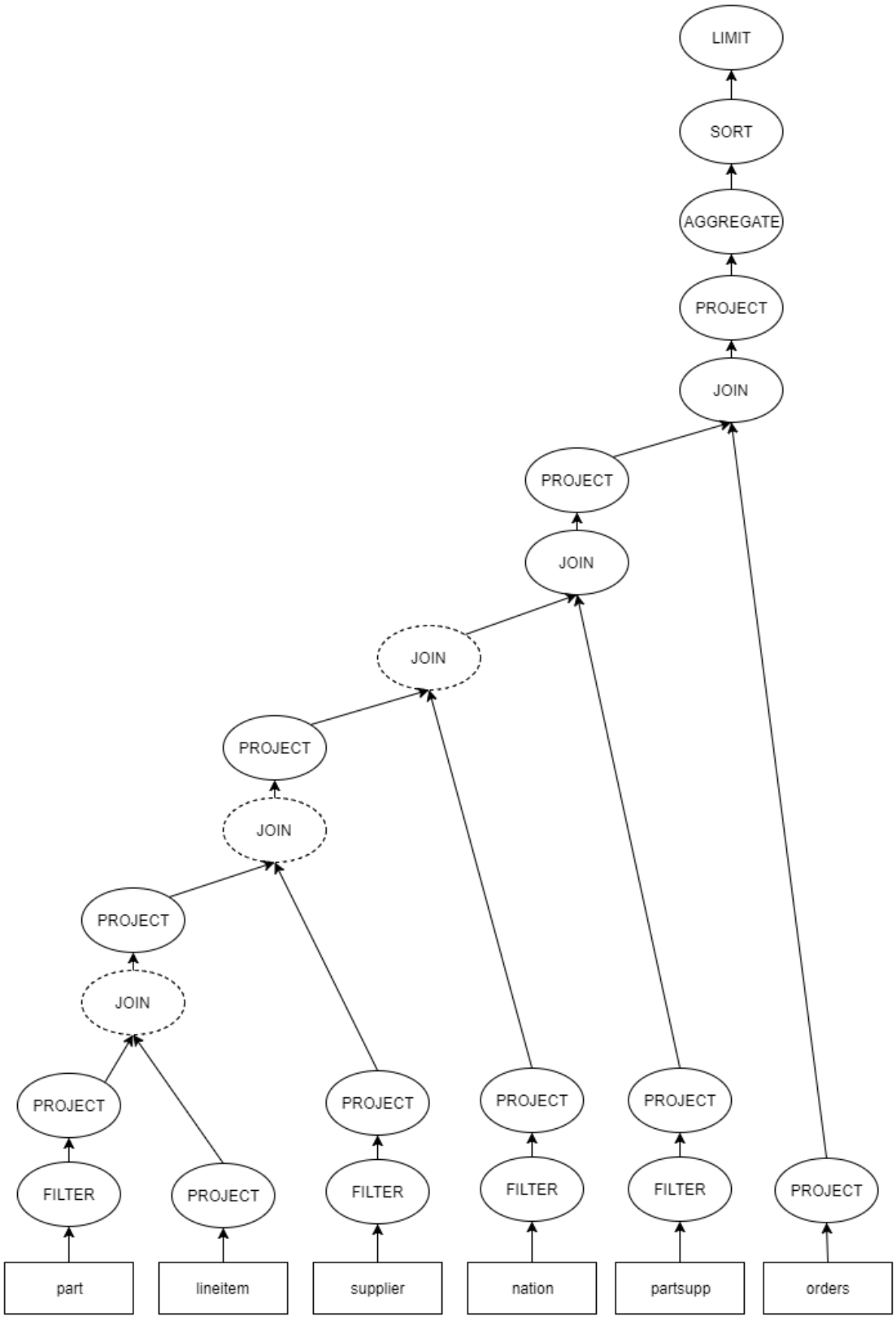


**Figure 4-17 Q9 Cost Based Optimizations Plan Batch 4**

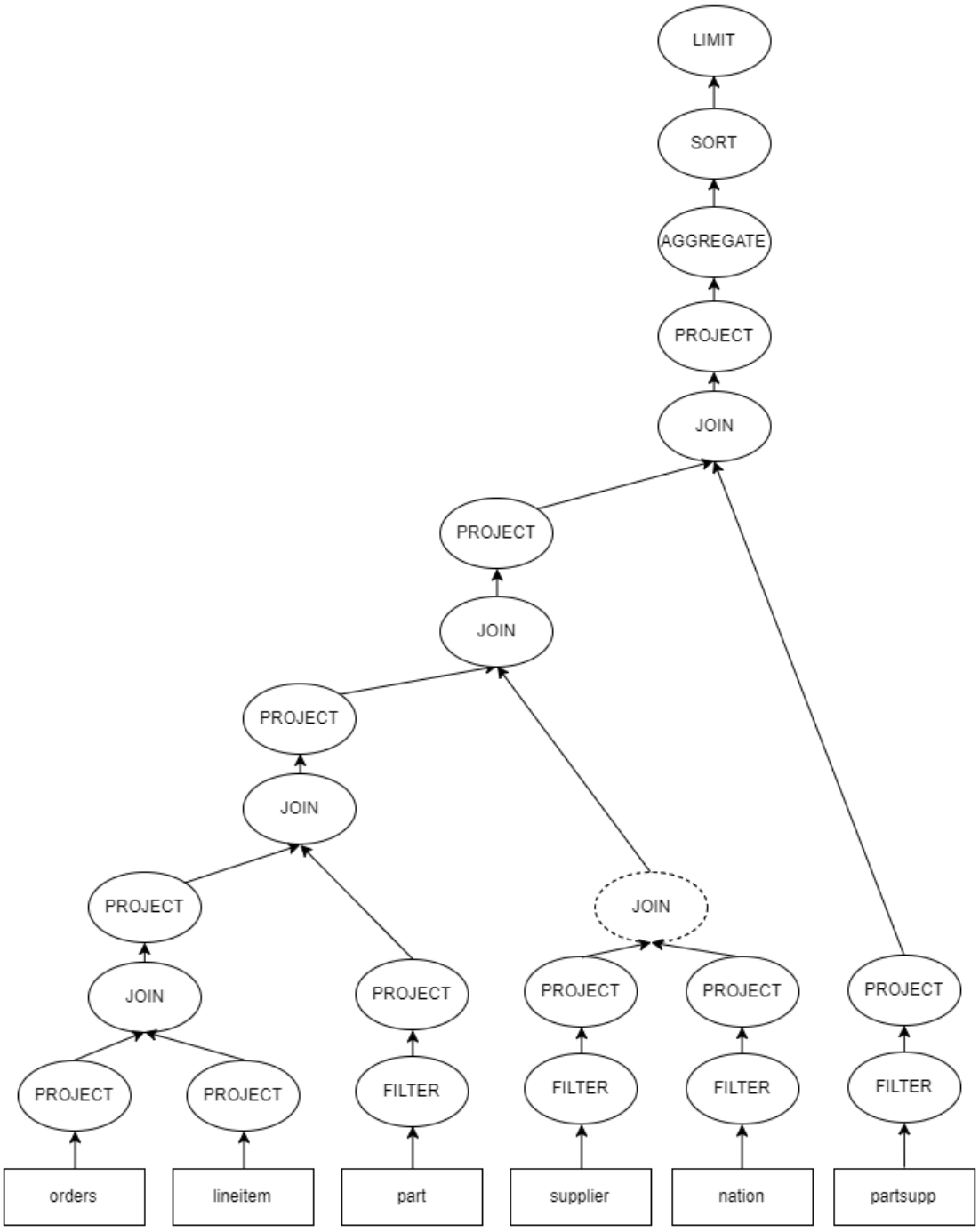


**Figure 4-18 Q9 Cost Based Optimizations with Basic Stats Plan (slow) Batch 1**

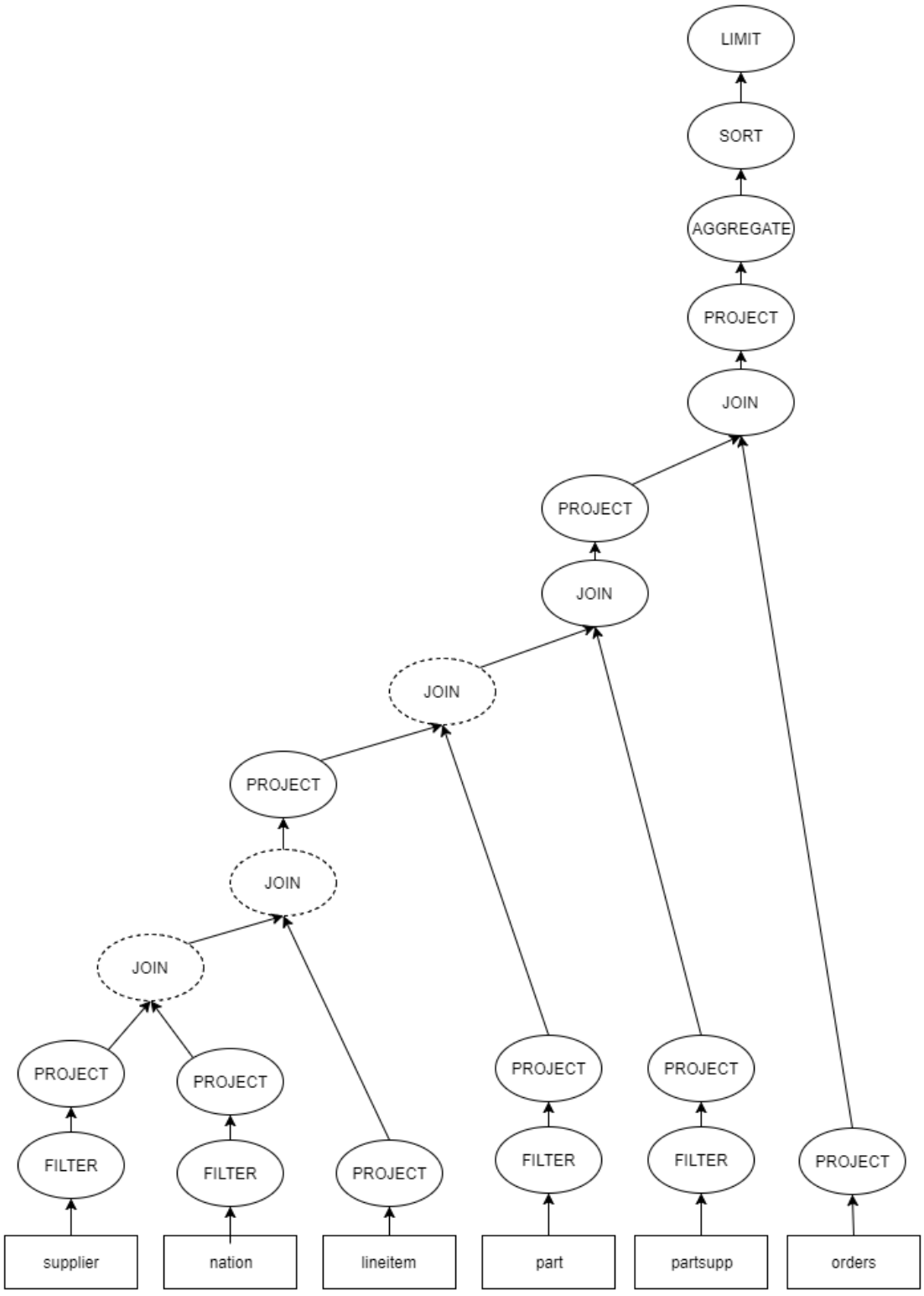


**Figure 4-19 Q9 Cost Based Optimizations with Stats from Sampling Plan Batch 6 (fast)**

**Q 21**

Batch sizes

| | Batch 1 | Batch 2 | Batch 3 | Batch 4 | Batch 5 | Batch 6 | Batch 7 |
|---|---|---|---|---|---|---|---|
| Lineitem | 1K | 9M | 200K | 7M | 8M | 5M | 9M |
| Orders | 1K | 8M | 7M | 1M | 600K | 5M | 9M |

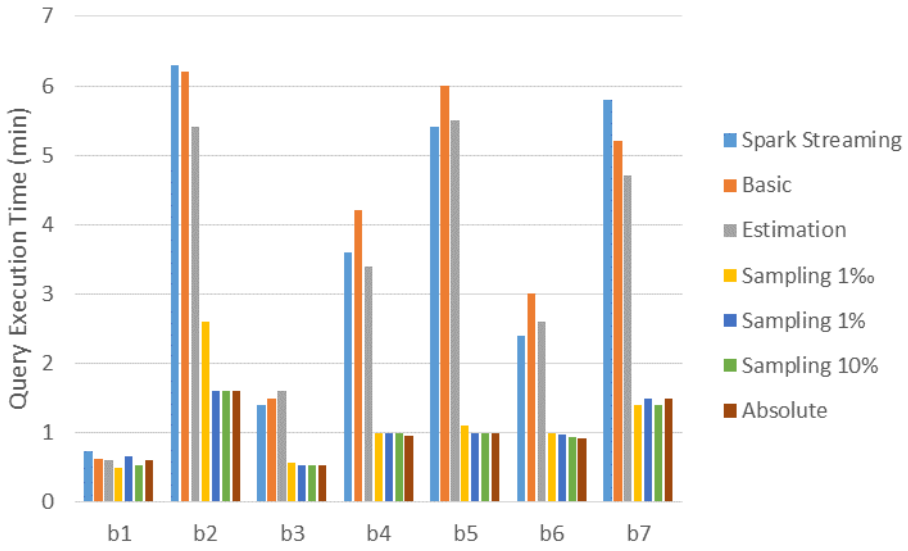


**Figure 4-20 Q21 Query Execution Time**

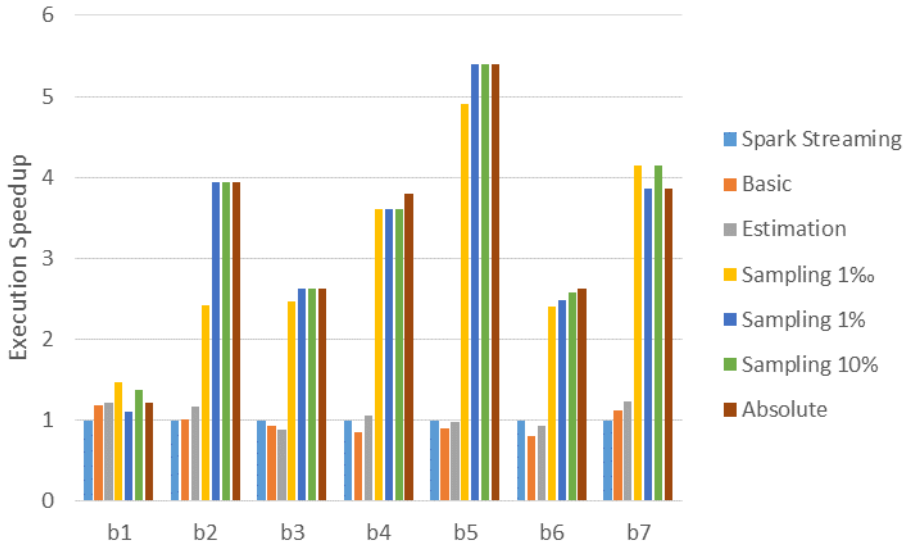


**Figure 4-21 Q21 Query Execution Speedup**

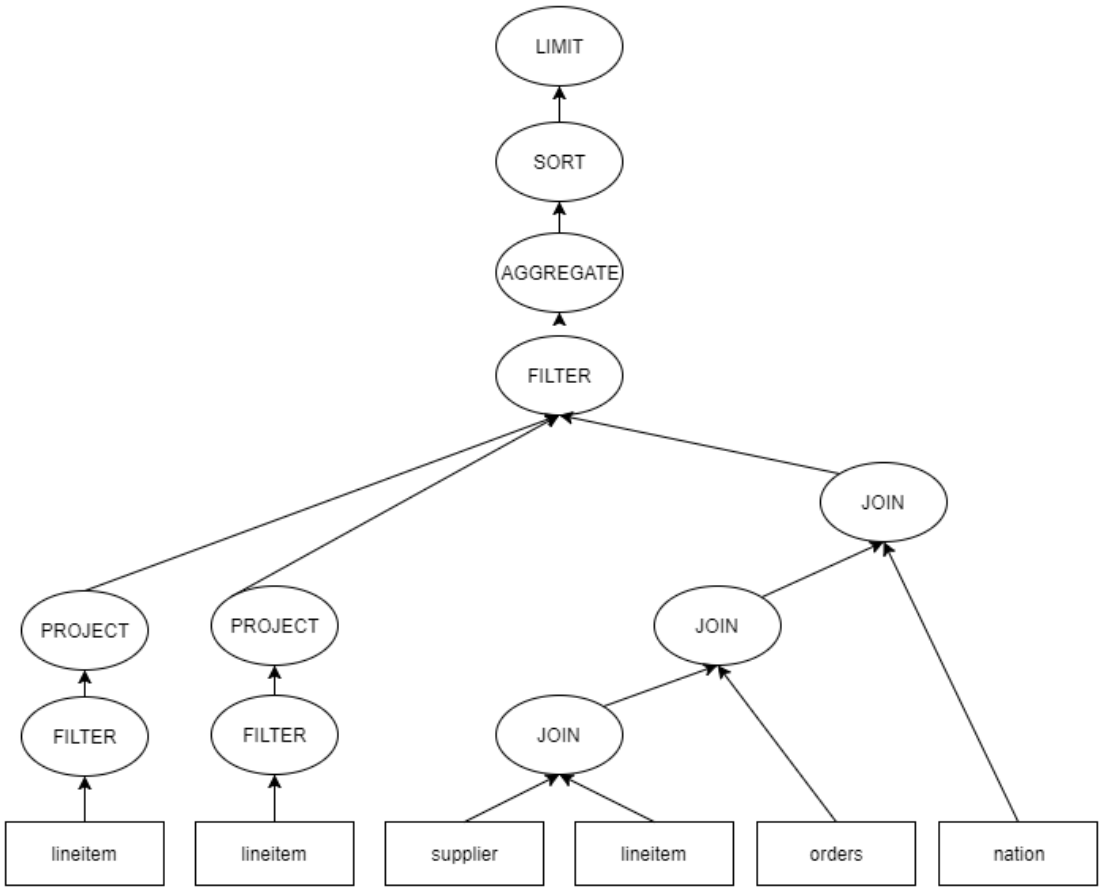


**Figure 4-22 Q21 Analyzed Plan**

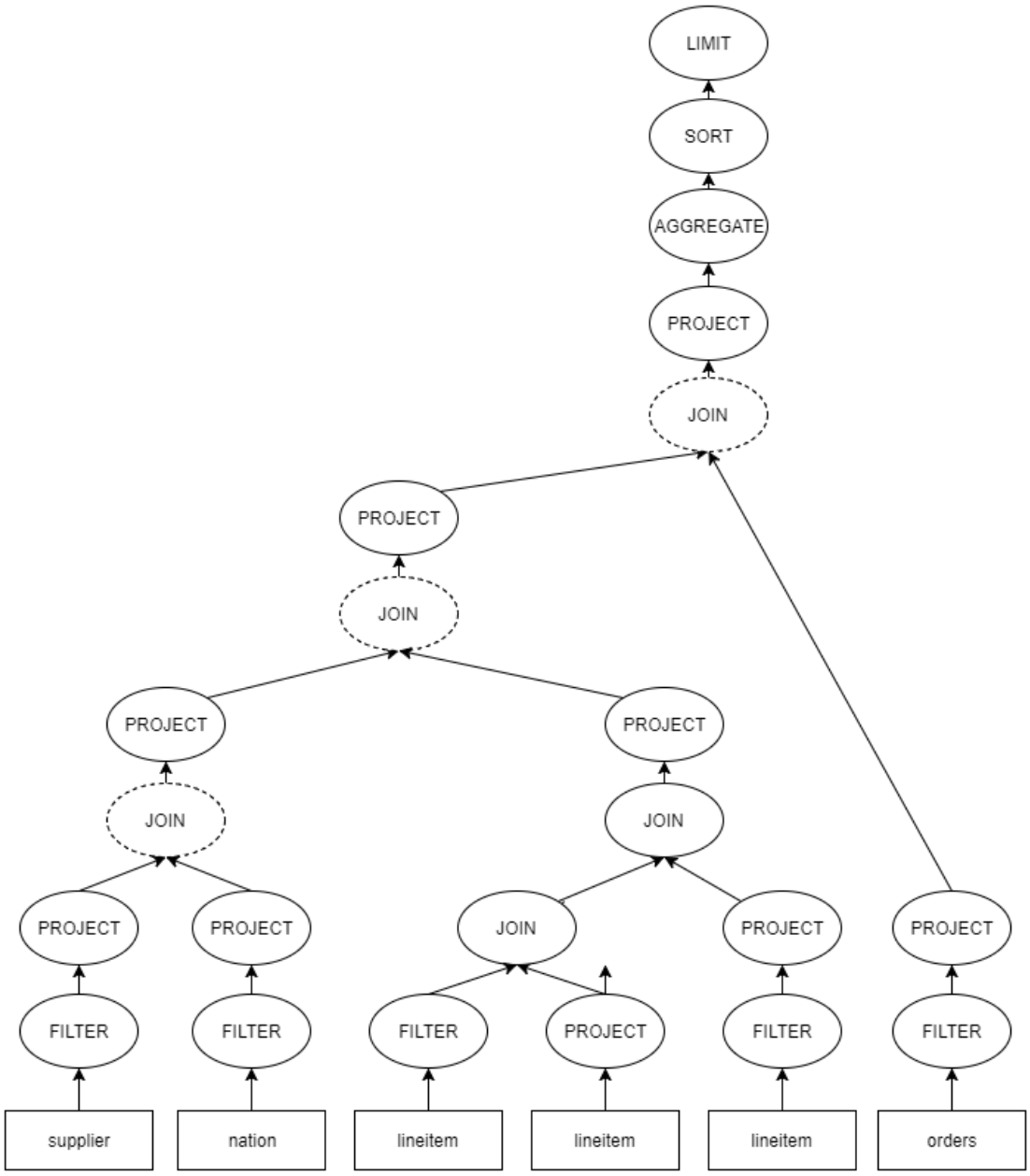


**Figure 4-23 Q21 Cost Based Optimizations Plan Batch 2**

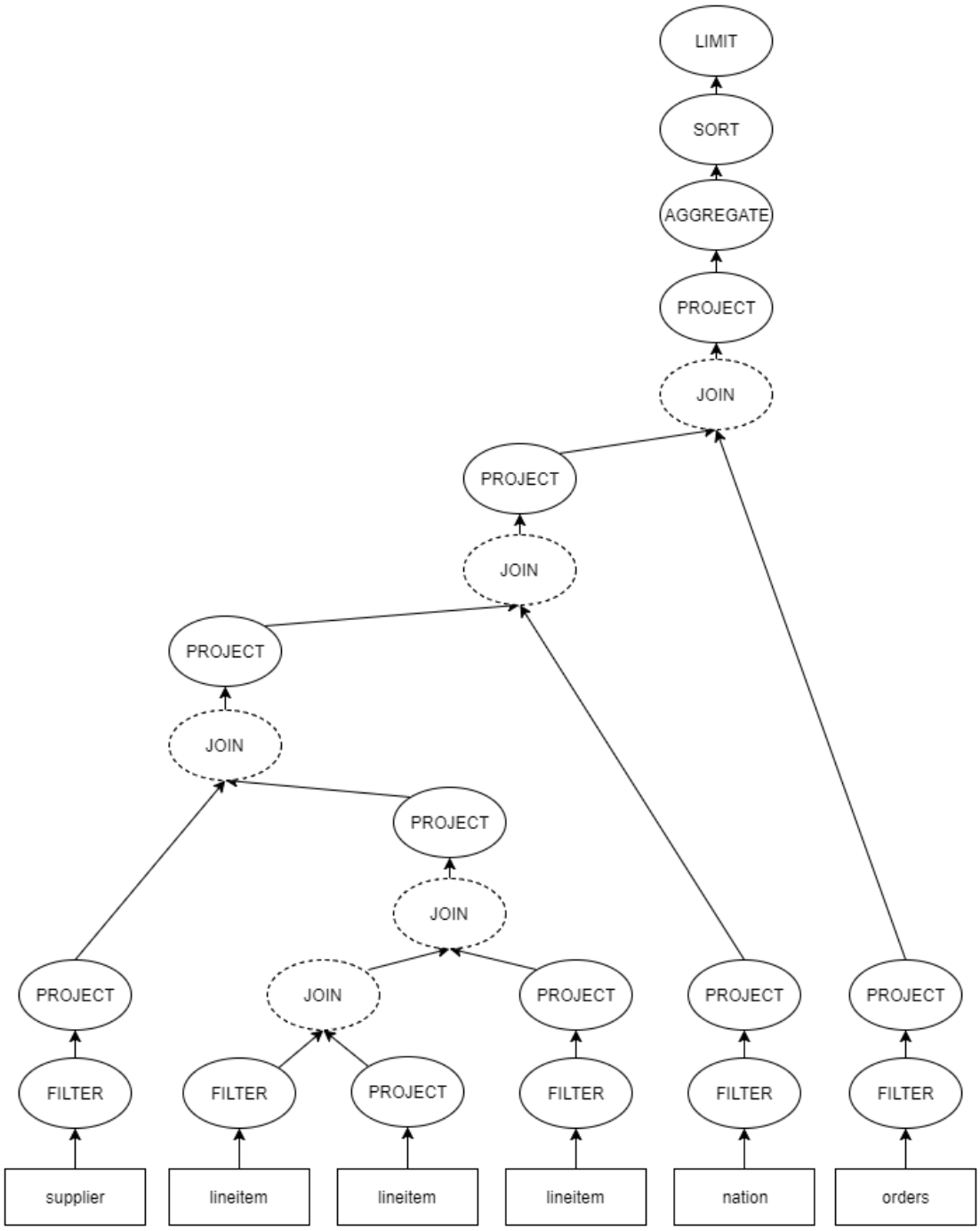


**Figure 4-24 Q21 Cost Based Optimizations Plan Batch 3**

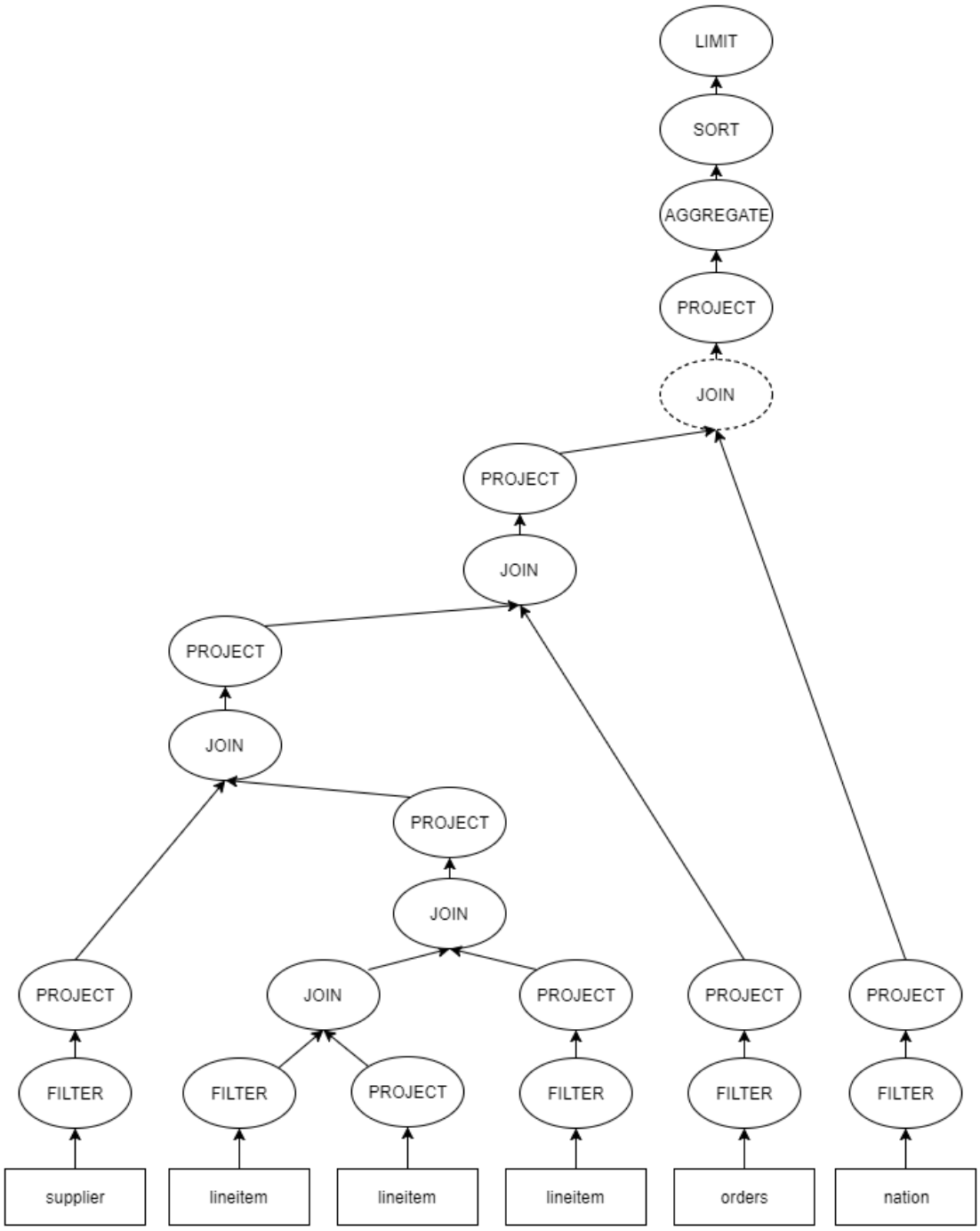


**Figure 4-25 Q21 Logical Optimizations Plan Batch 6**

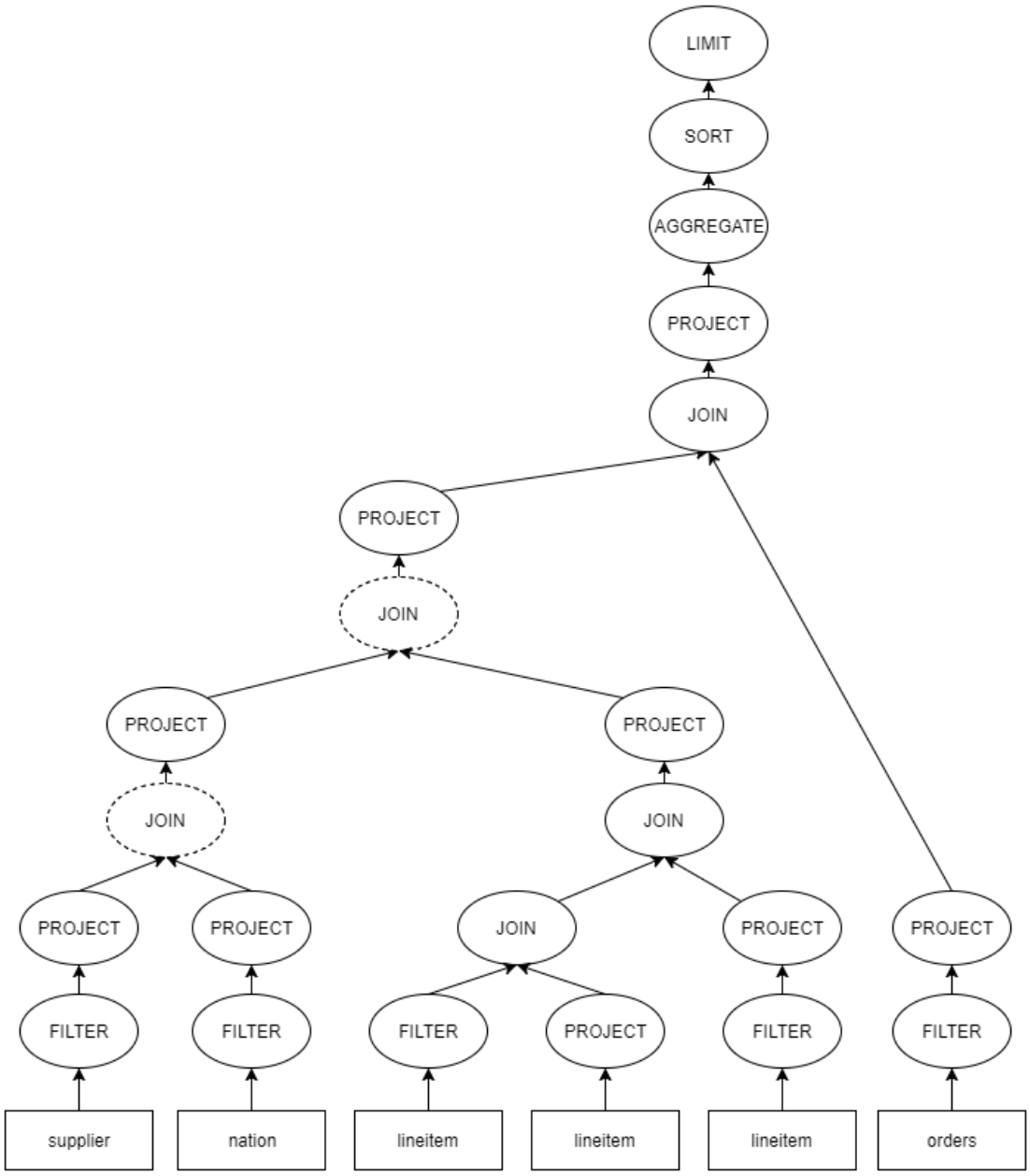


**Figure 4-26 Q21 Cost Based Optimizations Plan Batch 7**

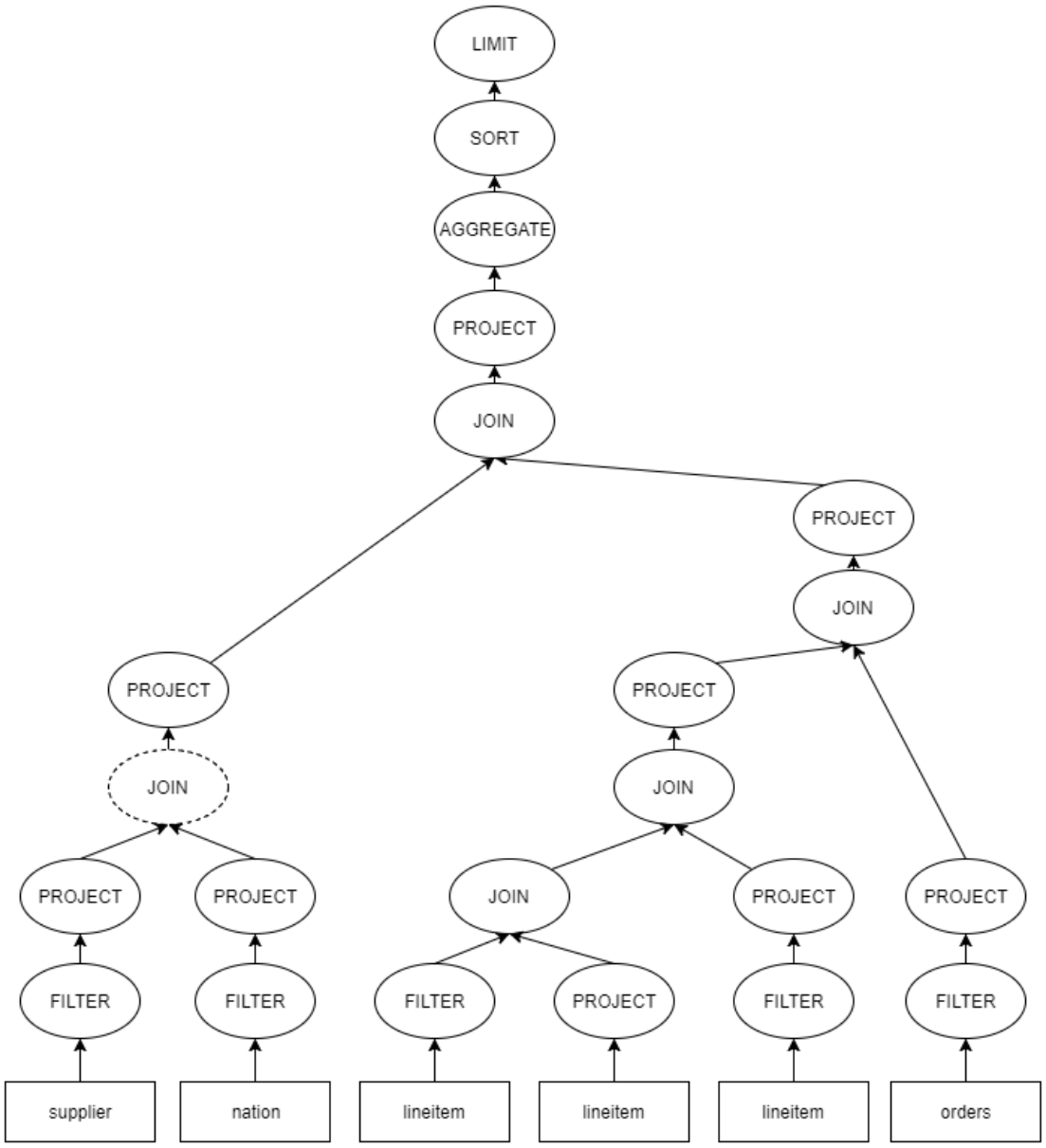


**Figure 4-27 Q21 Cost Based Optimizations with Basic Stats Plan**

## *4.5 Ανάλυση Αποτελεσμάτων*

Η πρώτη παρατήρηση επί των αποτελεσμάτων είναι πως οι βελτιστοποιήσεις κόστους που προκύπτουν είτε με χρήση απόλυτα σωστών στατιστικών είτε με χρήση στατιστικών από δειγματοληψία, παρουσιάζουν σταθερά καλύτερη απόδοση από αυτές που προκύπτουν από απλά στατιστικά (μέγεθος πινάκων και πλήθος εγγραφών) και από στατιστικά που εξάγονται με απλοϊκούς υπολογισμούς από ιστορικά δεδομένα.

Αυτό το συμπέρασμα ήταν αναμενόμενο, καθώς όπως αναλύθηκε στον τρόπο λειτουργίας του Catalyst, προκειμένου να υπολογιστεί με σχετική ακρίβεια το κόστος μιας συνένωσης, είναι σημαντικό να έχουμε όσο το δυνατόν πιο ακριβή στατιστικά.

Με χρήση απλοϊκών στατιστικών, έχουμε πλήρη έλλειψη του πλήθους διακριτών τιμών ανά στήλη και των μέγιστων/ελάχιστων τιμών. Αυτό έχει ως αποτέλεσμα όλες οι εκτιμήσεις κόστους του spark να αστοχούν σε μεγάλο βαθμό. Οι εκτιμήσεις στατιστικών στα φίλτρα δεν παράγουν νέα στατιστικά και διατηρούν τα στατιστικά του αρχικού πίνακα, οι εκτιμήσεις μεγεθών στις συνενώσεις θεωρούν πως το αποτέλεσμα έχει το μέγεθος καρτεσιανού γινομένου, κ.ο.κ. Συνεπώς, η χρήση του μεγέθους των πινάκων και του πλήθους των σειρών ως μοναδικά στατιστικά, αν και επιτυγχάνει την ενεργοποίηση και εφαρμογή βελτιστοποιήσεων κόστους, δεν οδηγεί σε ικανοποιητικά αποτελέσματα, έχει απρόβλεπτη συμπεριφορά και ορισμένες φορές οδηγεί και σε χειρότερα αποτελέσματα.

Όμοια, η χρήση έμπλουτισμένων στατιστικών που όμως προκύπτουν με βάση τα απλοϊκά στατιστικά και εκτίμηση των υπόλοιπων μεγεθών με βάση ιστορικά στατιστικά, δεν οδηγεί σε αποδεκτή βελτίωση. Καθώς η ακρίβεια των παρεχόμενων στατιστικών είναι μικρή, οδηγούμαστε πάλι σε απρόβλεπτη συμπεριφορά.

Μια ακόμα σημαντική παρατήρηση επί των αποτελεσμάτων είναι πως τα αποτελέσματα με χρήση στατιστικών 1‰, 1%, 10% και 100% ως επι το πλείστον συμβαδίζουν. Αυτό οφείλεται, σε μεγάλο βαθμό, στην υψηλή ομοιομορφία των δεδομένων που επιλέχθηκαν για τη διεξαγωγή της πειραματικής αξιολόγησης. Χάρη στην υψηλή ομοιομορφία, τα στατιστικά που προκύπτουν, ακόμα και με χαμηλή δειγματοληψία, έχουν υψηλή ακρίβεια. Αν γίνει επανάληψη των πειραμάτων με πιο ετερογενή δεδομένα, πιθανότατα τα αποτελέσματα θα είναι αρκετά διαφορετικά.

Στα αποτελέσματα με χρήση στατιστικών δειγματολειψίας, δεν παρατηρούμε μεγάλες αποκλίσεις. Μεταξύ των σχετικά μικρών αποκλίσεων, τις μεγαλύτερες διαφοροποιήσεις τις παρουσιάζει η δειγματοληψία 1‰, όπως είναι πάλι αναμενόμενο, λόγω της μικρότερης ακρίβειας. Μια άξιοσημείωτη παρέκλιση είναι στο Q9, στην 4$^{η}$ και 5$^{η}$ παρτίδα, όπου τα αποτελέσματα δειγματοληψίας 1‰ παρουσιάζουν το μεγαλύτερο speedup. Θα περίμενε κανείς πως το μεγαλύτερο speedup θα παρουσιαζόταν πάντα όταν κάναμε χρήση των απόλυτων στατιστικών, επομένως ένα τέτοιο αποτέλεσμα, εκ πρώτης όψεως, φαίνεται παράξενο. Κατόπιν περαιτέρω ανάλυσης όμως, προκύπτει πως η επιπρόσθετη επιτάχυνση προέκυψε από την μετατροπή μιας συνένωσης από Sort Merge Join σε Broadcast Join. Συγκεκριμένα, η μετατροπή αυτή έγινε αυτόματα, καθώς υπολογίστηκε πως ο ένας εκ των δυο πινάκων της ένωσης είχε μέγεθος μιρκότερο των 100MB, το οποίο και είναι το όριο αυτόματης συνένωσης με Broadcast Join (autoBroadcastJoinThreshold), ενώ στην πραγματικότητα το μέγεθος του πίνακα ήταν 260MB. Στην περίπτωσή μας, δεν

αντιμετωπίσαμε κάποιο πρόβλημα, καθώς είχαμε θέσει πολύ συντηρητική τιμή στο autoBroadcastJoinThreshold. Σε περίπτωση όμως που είχαμε θέσει το όριο πιο ψηλά, ώστε να εκμεταλλευτούμε πλήρως τις δυνατότητες του συστήματός μας, μια τέτοια λάθος εκτίμηση και απόπειρα Broadcast Join, με πίνακες απαγορευτικού μεγέθους, θα οδηγούσε σε αποτυχία ολοκλήρωσης του ερωτήματος.

Επί της προσαρμοστικής βελτιστοποίησης, παρατηρούμε εύκολα στα γραφήματα των πλάνων πως, όπως περιμέναμε, καθώς αλλάζουν τα μεγέθη των παρτίδων των ροών, αλλάζει διαρκώς και το παραγόμενο πλάνο. Αξιοποιώντας διαρκώς τα νέα στατιστικά, από παρτίδα σε παρτίδα αλλάζει η σειρά των συνενώσεων και ανάλογα με τα προβλεπόμενα μεγέθη αλλάζουν και οι αλγόριθμοι συνένωσης από Sort Merge Join σε Broadcast Join και αντίστροφα. Έτσι, επιτυγχάνεται συνεχής βελτίωση που ακολουθεί τις διαρκείς αλλαγές μεγέθους και μορφής των ροών δεδομένων.

# 5

# *Συμπεράσματα και Μελλοντικές Επεκτάσεις*

## *5.1 Συμπεράσματα*

Σε αυτή την διπλωματική εργασία παρουσιάστηκε σύστημα ανάλυσης και επεξεργασίας ροών δεδομένων, με προσαρμοστική βελτιστοποίηση εκτέλεσης ερωτημάτων. Τα σενάρια που εξετάστηκαν από αυτό το σύστημα ήταν ερωτήματα συνένωσης ροών δεδομένων με στατικά δεδομένα, χρησιμοποιώντας ως dataset το benchmark λήψης αποφάσεων TPC-H, το οποίο είναι γνωστό για τα απαιτητικά ερωτήματά του με πολλαπλές συνενώσεις.

Προκειμένου να επιτευχθεί τόσο κλιμακωσιμότητα όσο και ανοχή σε σφάλματα, χρησιμοποιήθηκαν κατανεμημένα συστήματα και τεχνολογίες όπως Spark με datastore HDFS και Kafka.

Το σύστημα είναι μια επέκταση του Spark Streaming, η οποία λαμβάνοντας υπόψιν στατιστικά για τα χαρακτηριστικά των ροών δεδομένων προς ανάλυση, παράγει διαρκώς πλάνο που ανταποκρίνεται στις αλλαγές τους.

Μέσω πειραμάτων, αξιολογήθηκε η απόδοση του συστήματος ανάλογα με το είδος και την ακρίβεια των στατιστικών βάσει των οποίων παράγεται διαρκώς το βελτιστοποιημένο πλάνο. Στα πειράματα αυτά διαπιστώσαμε, όπως ήταν αναμενόμενο, πως το σύστημα λειτουργεί βέλτιστα με αναλυτικά, πλήρη και ακριβή στατιστικά των ροών. Παρατηρήθηκε όμως εξίσου ικανοποιητική απόδοση και με στατιστικά που προκύπτουν από δειγματοληψία της ροής (έως και 1‰), αν και ενδέχεται η υψηλή απόδοση με χρήση αυτών των στατιστικών να βασίζεται

σε μεγάλο βαθμό και στην επιλογή του Dataset και την ομοιομορφία των δεδομένων του. Αντίθετα, η χρήση βασικών στατιστικών - το πλήθος τον μηνυμάτων και το συνολικό μέγεθος της ροής – και η χρήση προσεγγιστικά υπολογισμένων αναλυτικών στατιστικών, με απλοϊκές μεθόδους, δεν οδηγούν σε αξιοσημείωτες βελτιώσεις. Δεν οδηγούν συχνά σε αλλαγή πλάνου εκτέλεσης και μερικές φορές μάλιστα καταλήγουν σε υποδεέστερο πλάνο του αρχικού.

## *5.2 Μελλοντικές Επεκτάσεις*

Ως επεκτάσεις της παρούσας διπλωματικής εργασίας, προτείνουμε τα εξής.

Αρχικά, θα ήταν χρήσιμη η περαιτέρω διερεύνηση της αποσύμπλεξης της εξαγωγής στατιστικών από την παραγωγή της ροής δεδομένων. Ένα απολύτως αυτόνομο σύστημα εξαγωγής στατιστικών, ως ξεχωριστό στάδιο προ-επεξεργασίας των δεδομένων, θα μπορέσει να μας δώσει μια πιο καθαρή εικόνα και του κόστους εξαγωγής των στατιστικών, ώστε να μπορέσει να εκτιμηθεί καλύτερα η αξία των επιπρόσθετων βελτιστοποιήσεων κόστους.

Επιπλέον, θα ήταν ενδιαφέρουσα η χρήση του συστήματος σε διαφορετικά datasets. Ενδιαφέρουσες εναλλακτικές είναι: 1) το JOB (Join Order Benchmark), το οποίο διαφοροποιείται από το επιλεγμένο στην έλλειψη πλήρους ομοιογένειας – που είναι έντονο χαρακτηριστικό του TPC-H και 2) το TPC-E benchmark, το οποίο περιλαμβάνει δεδομένα συναλλαγών που προσφέρονται για αναπαράσταση σε ροές δεδομένων.

Τέλος, πολύ ενδιαφέρουσα επέκταση θα ήταν η επέκταση του Catalyst, ώστε να αναγνωρίζει πότε λειτουργεί σε streaming context, να μπορεί να εντοπίζει ποια DataFrames ανήκουν σε ροές και ποια σε στατικά δεδομένα, στη συνέχεια, αναδιατάσσοντας τις συνενώσεις να φτιάχνει όσο το δυνατόν μεγαλύτερες ενδιάμεσες συστάδες από συνενώσεις στατικών δεδομένων και να τα αποθηκεύει προσωρινά για χρήση στο επόμενο batch. Στη συνέχεια, θα μπορούσε να εξετάζει ανάλογα με το πλήθος των ροών, τα στατιστικά τους και το μέγεθος των στατικών, προσωρινά αποθηκευμένων αποτελεσμάτων, αν συμφέρει περισσότερο η εκτέλεση πλάνου με τα προϋπολογισμένα στατικά αποτελέσματα ή η παραγωγή νέου πλάνου βάσει των νέων στατιστικών των ροών δεδομένων.

# 6

# *Βιβλιογραφία*


[1] “The world beyond batch: Streaming 101”

https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

[2] “The world beyond batch: Streaming 102”

https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

[3] “Apache Spark” https://spark.apache.org/

[4] “Apache Flink” https://flink.apache.org/

[5] “Apache Storm” https://storm.apache.org/

[6] “Apache Parquet” https://parquet.apache.org/

[7] “Apache Kafka” https://kafka.apache.org/

[8] “Redis” https://redis.io/

[9] "Openstack", https://www.openstack.org/

[10] “Kafka Python client” https://github.com/confluentinc/confluent-kafka-python

[11] “TPC-E” http://www.tpc.org/tpce/default.asp

[12] “TPC-DS” http://www.tpc.org/tpcds/default.asp

[13] “TPC-H” http://www.tpc.org/tpch/default.asp

[14] “Hive Analyze Table” https://docs.databricks.com/spark/latest/spark-sql/language-manual/analyze-table.html

[15] “Spark SerializeFromObject” https://github.com/apache/spark/blob/branch-2.2/sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/plans/logical/object.scala#L89

[16] “Why we need SQL for data stream processing and real-time streaming analytics” http://sqlstream.com/2017/02/sql-for-real-time-streaming-analytics/

[17] “Why We need SQL like Query Language for Realtime Streaming Analytics?” http://srinathsview.blogspot.gr/2015/02/why-we-need-sql-like-query-language-for.html

[18] “Deep Dive into Spark SQL’s Catalyst Optimizer” https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

[19] “Cost-Based Optimizer Framework for Spark SQL: Spark summit East talk by Ron Hu and Zhenhua Wang” https://www.slideshare.net/SparkSummit/costbased-optimizer-framework-for-spark-sql-spark-summit-east-talk-by-ron-hu-and-zhenhua-wang

[20] “Cost-Based Optimizer in Apache Spark 2.2” https://www.slideshare.net/databricks/costbased-optimizer-in-apache-spark-22

[21] “Scala Quasiquotes” http://docs.scala-lang.org/overviews/quasiquotes/intro.html

[22] “Spark Streaming Kafka Integration” https://spark.apache.org/docs/2.1.0/streaming-kafka-0-10-integration.html

[23] “Improvements to Kafka integration of Spark Streaming” https://databricks.com/blog/2015/03/30/improvements-to-kafka-integration-of-spark-streaming.html

[24] R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. In Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data, pages 261–272, May 2000.

[25] Stratis D. Viglas , Jeffrey F. Naughton, Rate-based query optimization for streaming information sources, Proceedings of the 2002 ACM SIGMOD international conference on Management of data, June 03-06, 2002, Madison, Wisconsin [doi>10.1145/564691.564697]

[26] Michael Armbrust, Reynold Xin, Cheng Lian, Yin Yuai, Davies Liu, Joseph Bradley, Xiangrui Meng, Tomer Kaftan, Michael Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational data processing in spark. In ACM Special Interest Group on Management of Data, 2015.

[27] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, Thomas Neumann. How Good Are Query Optimizers, Really?. In Proceedings of the VLDB Endowment, Vol. 9, No. 3

[28] Y. Chabchoub, G. Hébrail, Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window, in IEEE International Conference on Data Mining Workshops, Sydney, Australia, December 2010.

[29] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing, pages 10--10. USENIX Association, 2012.

[30] Goetz Graefe , William J. McKenna, The Volcano Optimizer Generator: Extensibility and Efficient Search, Proceedings of the Ninth International Conference on Data Engineering, p.209-218, April 19-23, 1993

[31] X. Ren and O. Curé, "Strider: A hybrid adaptive distributed RDF stream processing engine," CoRR, vol. abs/1705.05688, 2017. [Online]. Available: http://arxiv.org/abs/1705.05688

[32] "Data size estimates" https://followthedata.wordpress.com/2014/06/24/data-size-estimates/

[33] "2.5 quintillion bytes of data created every day. How does CPG & Retail manage it?" https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/

[34] "Google's Datacenters on Punch Cards" https://what-if.xkcd.com/63/

[35] S. Heule, M. Nunkesser, and A. Hall. HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. EDBT, 2013.