ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

Βελτιστοποιημένη Αποθήκευση και Διαχείριση Δεδομένων:

Apache Parquet και Delta Tables

Τμήμα Ψηφιακών Συστημάτων: Διπλωματική Εργασία

ΠΜΣ Πληροφοριακά Συστήματα και Υπηρεσίες

Ειδίκευση: «Προηγμένα Πληροφοριακά Συστήματα»

ΕΜΜΑΝΟΥΗΛ ΖΑΧΑΡΙΟΥΔΑΚΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΧΡΗΣΤΟΣ ΔΟΥΛΚΕΡΙΔΗΣ

Πειραιάς 2023-2024

Περιεχόμενα

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στο Πανεπιστήμιο Πειραιώς στο Μεταπτυχιακό Τμήμα Πληροφοριακά Συστήματα και Υπηρεσίες εν έτη 2024.

Θα ήθελα να ευχαριστήσω σε αυτό το σημείο τον επιβλέποντα καθηγητή μου Χρήστο Δουλκερίδη, Αναπληρωτή Καθηγητή στο Τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς για δύο πολύ σημαντικούς λόγους, αρχικά καθώς έπειτα από προτροπή μου τροποποιήσαμε κατά ένα μέρος το θέμα της εργασίας για να ταιριάζει με το εργασιακό μου αντικείμενο βοηθώντας με έτσι να εμβαθύνω σε αυτό, πράγμα πολύ σημαντικό για εμένα αλλά μετέπειτα και για την πολύτιμη καθοδήγηση που μου προσέφερε και το χρόνο που διέθεσε δίνοντάς μου χρήσιμες συμβουλές και οδηγίες για την ολοκλήρωση της πτυχιακής μου εργασίας. Χωρίς την υποστήριξη του θα ήταν αδύνατη η ολοκλήρωση της εργασίας.

Στο ίδιο πλαίσιο ευγνωμοσύνης, θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματος για τη συμβολή τους στην επιστημονική και τεχνολογική μου συγκρότηση στα χρόνια της μεταπτυχιακής φοίτησής μου.

Κεφάλαιο 1: Εισαγωγή

1.1 Αντικείμενο Διπλωματικής Εργασίας

Το αντικείμενο με το οποίο ασχολείται η διπλωματική εργασία είναι η συγκριτική ανάλυση του μορφής των Parquet αρχείων έναντι των CSV στην απόδοση της αποθήκευσης των δεδομένων στο εσωτερικό τους.

Χρησιμοποιώντας κατάλληλες μετρικές, η εργασία διερευνά τις πτυχές όπως η επίδοση στην ανάγνωση και εγγραφή δεδομένων, ο χρόνος ανάκτησης, η αποτελεσματικότητα της συμπίεσης και η χρήση του διαθέσιμου χώρου.

Μέσω μιας σειράς πειραμάτων και αναλύσεων, παρουσιάζονται τα οφέλη των Parquet αρχείων έναντι των παραδοσιακών CSV. Η μελέτη αυτή αποτελεί σημαντική προσθήκη στον τομέα της αποθήκευσης δεδομένων, παρέχοντας πολύτιμες πληροφορίες για την επιλογή της κατάλληλης μορφής αρχείων.

Εν συνεχεία, γίνεται μία εκτενής περιγραφή και ανάλυση των Delta Tables, ως προς το πως αποθηκεύουν και διαχειρίζονται τα Parquet αλλά και τα log αρχεία στην τοποθεσία τους που είναι αποθηκευμένα.

Και σε αυτό το μέρος θα γίνεται προσπάθεια μέσω διαφόρων τεχνικών να βελτιωθεί η απόδοση των συγκεκριμένων πινάκων μειώνοντας τον χρόνο απόκρισης σε περίπλοκα ερωτήματα.

Όλα τα παραπάνω προϋποθέτουν σε βάθος κατανόηση των Delta Tables τα οποία και είναι μια επέκταση των Parquet αρχείων που θα αναλυθούν στο πρώτο μέρος της διπλωματικής.

1.2 Δομή Διπλωματικής Εργασίας

Το **1ο Κεφάλαιο** ασχολείται με μια εισαγωγική αναφορά-παρουσίαση για το αντικείμενο της διπλωματικής εργασίας και τον σκοπό της.

Στο **2ο Κεφάλαιο** γίνεται αναφορά βασικών πληροφοριών που απαιτούνται για την κατανόηση των Parquet αρχείων.

Στο **3ο Κεφάλαιο** υπάρχουν πληροφορίες όσο αφορά τον κώδικα που έχει υλοποιηθεί για τους σκοπούς της εργασίας.

Στο **4ο Κεφάλαιο** βρίσκονται τα αποτελέσματα των πειραμάτων καθώς και το συμπέρασμα που πηγάζουν από εκείνα.

Το **5ο Κεφάλαιο** ασχολείται με βασικές εισαγωγικές πληροφορίες πάνω στους Delta Tables, ώστε να μπορέσει γίνει μια αρχική κατανόηση του τι ακριβώς είναι και γιατί καθίσταται σημαντική η αξιοποίηση του συγκεκριμένου τρόπου αποθήκευσης.

Στο **6ο Κεφάλαιο** υπάρχει με την αναλυτική περιγραφή του τρόπου λειτουργίας των Delta Tables για κάθε Transaction (συναλλαγή) που μπορούν να υποστούν τα δεδομένα που είναι αποθηκευμένα μέσα σε αυτά.

Στο **7ο Κεφάλαιο** γίνεται αναφορά των πιο βασικών τρόπων βελτιστοποίησης των συγκεκριμένων πινάκων καθώς και η εφαρμογή μερικών εξ' αυτών πάνω σε ένα περίπλοκο ερώτημα μαζί με τα αποτελέσματα και το συμπέρασμα που αντλείται μέσα από το παραπάνω.

1.3 Σκοπός Εργασίας

Σκοπός της εργασίας είναι μέσα από πολλά και διαφορετικά select queries που θα πραγματοποιηθούν μέσα από το πρόγραμμα μας, να συλλεχθούν οι κατάλληλες πληροφορίες, ώστε να αποδειχθεί, ότι πράγματι ένα Parquet αρχείο μπορεί να αποτελεί πιο αποδοτικό τρόπο αποθήκευσης δεδομένων, εν αντίθεση με ένα CSV.

Η αποτύπωση των αποτελεσμάτων σε διαγραμματική μορφή της κάθε επανάληψης ξεχωριστά θα βοηθήσει να παραχθούν τα κατάλληλα συμπεράσματα.

Το report που θα υλοποιηθεί, χωρίζεται σε δύο κύριες κατηγορίες:

- Μία αφιερωμένη στα αποτελέσματα που από το CSV αρχείο.
- Μία αφιερωμένη στα αποτελέσματα που από τα Parquet αρχεία.

Θα γίνει εξέταση για κάθε αρχείο ως προς:

- Τον χρόνο που απαιτείται για την ανάγνωση του αρχείου
- Τον χρόνο εκτέλεσης για ένα συγκεκριμένο ερώτημα δεδομένων
- Τη χρήση μνήμης και το συνολικό μέγεθος αρχείου.

Συγκρίνοντας αυτές τις μετρήσεις, μπορεί να αντληθεί και να εξακριβωθεί το συμπέρασμα για το ποια μορφή αποθήκευσης είναι πιο αποδοτική. Σκοπός μας είναι να εκτελεστούν ορισμένα ερωτήματα (θα διευκρινιστούν στην συνέχεια) πάνω στα CSV και Parquet αρχεία, ώστε να εξαχθούν ορισμένα συμπεράσματα, ως προς την αποδοτικότητα και την αποτελεσματικότητα των αρχείων. Μέσω αυτής της υλοποίησης θα επιβεβαιώσουμε πως τα Parquet αρχεία είναι πιο αποδοτικά σε σχέση με τα CSV, ως προς τα resources που καταναλώνουν, ως προς την ταχύτητα με την απαντούν πάνω στα ερωτήματα μας.

Κεφάλαιο 2: Parquet

2.1 Ορισμός parquet

Το Apache Parquet είναι μια μορφή αρχείου ανοιχτού κώδικα που αποθηκεύει δεδομένα σε μορφή στήλης, δηλαδή αποθηκεύει τις τιμές κάθε στήλης μαζί, αντί να αποθηκεύει ολόκληρες σειρές διαδοχικά. Ως μορφή στηλών αποθήκευσης δεδομένων, προσφέρει πολλά πλεονεκτήματα έναντι των μορφών που βασίζονται σε γραμμές, για αναλυτικούς φόρτους εργασίας. Η επιλογή της μορφής δεδομένων μπορεί να έχει σημαντικές επιπτώσεις για την απόδοση και το κόστος των ερωτημάτων, επομένως είναι σημαντικό να γίνονται αντιληπτές οι διαφορές μεταξύ του Apache Parquet και άλλων μορφών αρχείων [όπως αναφέρεται στο 1].

2.2 Βασικά Χαρακτηριστικά του parquet

Η μορφή του αρχείου Parquet ξεχωρίζει για τις ιδιαίτερες ιδιότητές του που αλλάζουν τον τρόπο δομής, συμπίεσης και χρήσης των δεδομένων. Σε αυτό το σημείο θα αναλυθούν τα κύρια χαρακτηριστικά που κάνουν το παρκέ διαφορετικό από τα κανονικά σχήματα.

Πιο αναλυτικά:

- Μορφή βάσει στήλης:

Η βασική διαφορά μεταξύ μιας μορφής αρχείου CSV και Parquet είναι ο διαφορετικός τρόπος οργάνωσης της πληροφορίας. Μια μορφή αρχείου Parquet είναι δομημένη κατά σειρά, με κάθε ξεχωριστή στήλη να είναι ανεξάρτητα προσβάσιμη από τις υπόλοιπες.

Δεδομένου ότι τα δεδομένα σε κάθε στήλη είναι του ίδιου τύπου, η μορφή αρχείου Parquet καθιστά δυνατή την κωδικοποίηση, τη συμπίεση και τη βελτιστοποίηση της αποθήκευσης δεδομένων.

- Δυαδική μορφή:

Οι μορφές αρχείων Parquet αποθηκεύουν δεδομένα σε δυαδική μορφή, γεγονός που μειώνει το γενικό κόστος της αναπαράστασης κειμένου. Αυτό καθιστά αδύνατον από τον χρήστη να διαβάσει την πληροφορία και τα δεδομένα που κρύβει μέσα του το αρχείο, απλά ανοίγοντας το, όπως στην περίπτωση του CSV.

- Συμπίεση δεδομένων:

Οι μορφές αρχείων Parquet υποστηρίζουν διάφορους αλγόριθμους συμπίεσης, όπως Snappy, Gzip και LZ4, με αποτέλεσμα να μειώνουν κατά πολύ το μέγεθος του αρχείου, σε σύγκριση με μη συμπιεσμένες μορφές όπως αυτή του CSV. Αυτό σημαίνει πως το Parquet αρχείο μπορεί να υποστεί μείωση μεγέθους σχεδόν 75%.

- Ενσωματωμένα μεταδεδομένα (meta-data):

Οι μορφές αρχείων Parquet περιλαμβάνουν μεταδεδομένα που παρέχουν πληροφορίες σχετικά με το σχήμα, τις ρυθμίσεις συμπίεσης, τον αριθμό των τιμών, τη θέση των στηλών, την ελάχιστη τιμή, τη μέγιστη τιμή, και τον τύπο κωδικοποίησης.

Τα ενσωματωμένα μεταδεδομένα βοηθούν την μηχανή στην αποτελεσματική ανάγνωση και επεξεργασία των δεδομένων. Οποιοδήποτε πρόγραμμα που χρησιμοποιείται για την ανάγνωση των δεδομένων μπορεί επίσης να έχει πρόσβαση στα μεταδεδομένα για να προσδιορίσει τον τύπο δεδομένων που αναμένεται να βρεθεί σε μια δεδομένη στήλη [όπως αναγράφεται στην βιλιογραφία 8].

2.3 Πλεονεκτήματα Parquet

Τα κύρια σημεία που δίνουν στο Parquet file format την δυνατότητα να ξεχωρίζει έναντι των υπόλοιπων τύπων αρχείων είναι το κόστος αποθήκευσης, το κόστος υπολογισμού και ανάλυσης των δεδομένων που περιέχει.

- Εξοικονόμηση κόστους αποθήκευσης:

Τα Parquet έχουν κατασκευαστεί για να υποστηρίζουν ευέλικτες επιλογές συμπίεσης και αποτελεσματικά σχήματα κωδικοποίησης. Τα δεδομένα μπορούν να συμπιεστούν χρησιμοποιώντας έναν από τους πολλούς διαθέσιμους κωδικοποιητές (όπως Snappy, Gzip και LZ4).
Επομένως, το Parquet είναι καλό για την αποθήκευση μεγάλων δεδομένων οποιουδήποτε είδους (δομημένοι πίνακες δεδομένων, εικόνες, βίντεο, έγγραφα).

- Εξοικονόμηση υπολογιστικού κόστους:

Καθώς ο τύπος δεδομένων για κάθε στήλη είναι ίδιος, η συμπίεση κάθε στήλης είναι απλή, κάνοντας τα ερωτήματα ακόμα πιο γρήγορα. Κατά την υποβολή των ερωτημάτων, παρακάμπτονται τα μη σχετικά δεδομένα πολύ γρήγορα. Ως αποτέλεσμα, τα ερωτήματα είναι λιγότερο χρονοβόρα σε σύγκριση με βάσεις δεδομένων που προσανατολίζονται σε γραμμές.

- Αποτελεσματική ανάλυση και συναλλαγές υψηλής ταχύτητας:

Τα αρχεία Parquet είναι κατάλληλα για περιπτώσεις χρήσης ηλεκτρονικής αναλυτικής επεξεργασίας (OLAP). Με το Parquet, υπάρχει γρήγορη πρόσβαση σε δεδομένα για συγκεκριμένες στήλες και βελτιωμένη απόδοση σε κατανεμημένα περιβάλλοντα επεξεργασίας δεδομένων.

Τα Parquet χρησιμοποιούνται συχνά σε συνδυασμό με τις παραδοσιακές βάσεις δεδομένων Online Transaction Processing (OLTP). Οι βάσεις δεδομένων OLTP είναι βελτιστοποιημένες για γρήγορες ενημερώσεις και εισαγωγή δεδομένων, ενώ το Parquet τις συμπληρώνει προσφέροντας αποτελεσματικές δυνατότητες ανάλυσης [όπως περιγράφεται στην βιβλιογραφία 2].

2.4 Διαφορές μεταξύ Parquet και CSV

Ενώ το CSV χρησιμοποιείται ευρέως σε μεγάλους οργανισμούς, οι μορφές αρχείων CSV και Parquet είναι κατάλληλες για διαφορετικές περιπτώσεις χρήσης.

Διαφορές μεταξύ των δύο αρχείων:

- Αποτελεσματικότητα αποθήκευσης:

Η μορφή αρχείου Parquet είναι μια μορφή στηλών αποθήκευσης, που σημαίνει ότι τα δεδομένα για κάθε στήλη αποθηκεύονται μαζί. Ο μηχανισμός αποθήκευσης επιτρέπει καλύτερη συμπίεση και συνήθως οδηγεί σε μικρότερα μεγέθη αρχείων σε σύγκριση με μορφές που βασίζονται σε σειρά.

Το CSV είναι μια μορφή που βασίζεται σε γραμμές, όπου κάθε σειρά αναπαρίσταται ως ξεχωριστή γραμμή στο αρχείο. Η μορφή δεν προσφέρει συμπίεση, με αποτέλεσμα το μέγεθος των αρχείων να είναι αρκετά μεγάλο.

- Απόδοση στα ερωτήματα:

Τα CSV χρειάζονται να διαβάσουν ολόκληρο το αρχείο για να υποβάλουν ερώτημα σε μία μόνο στήλη, κάτι που είναι εξαιρετικά αναποτελεσματικό.

Από την άλλη πλευρά, η αποθήκευση στη στήλη και η αποτελεσματική συμπίεση του Parquet το καθιστούν κατάλληλο για αναλυτικά ερωτήματα που χρειάζονται μόνο πρόσβαση σε συγκεκριμένες

στήλες. Αυτή η συμπίεση οδηγεί σε ταχύτερη απόδοση ερωτημάτων όταν ασχολούμαστε με μεγάλα σύνολα δεδομένων.

- Εξέλιξη σχήματος:

Η μορφή αρχείου Parquet υποστηρίζει την εξέλιξη του σχήματος. Πιο συγκεκριμένα, δίνεται η δυνατότητα να προστίθενται νέες στήλες δεδομένων χωρίς να χρειάζεται να λαμβάνεται υπόψη για το υπάρχον σύνολο δεδομένων μας (φαινόμενο αρκετά συχνό κυρίως σε Extract Transform Load διαδικασίες, όπου το schema του Target πίνακα μπορεί να αλλάξει).
Τα αρχεία CSV από την άλλη πλευρά, δεν υποστηρίζουν εγγενώς την εξέλιξη σχήματος, κάτι που μπορεί να είναι περιορισμός εάν το σχήμα δεδομένων σας αλλάζει συχνά.

- Συμβατότητα και χρηστικότητα:

Το Parquet είναι σχεδιασμένο για μηχανές και όχι για ανθρώπους. Η χρήση καθώς και η ανάγνωση του απαιτεί πρόσθετες. Το CSV από την άλλη, είναι μια απλή και ευρέως υποστηριζόμενη μορφή που μπορεί εύκολα να διαβαστεί και να γραφτεί από ανθρώπους και σχεδόν οποιοδήποτε εργαλείο επεξεργασίας δεδομένων ή γλώσσα προγραμματισμού, καθιστώντας το πολύ ευέλικτο και προσβάσιμο [όπως αναδεικνύεται στην βιβλιογραφία 3 και 7].

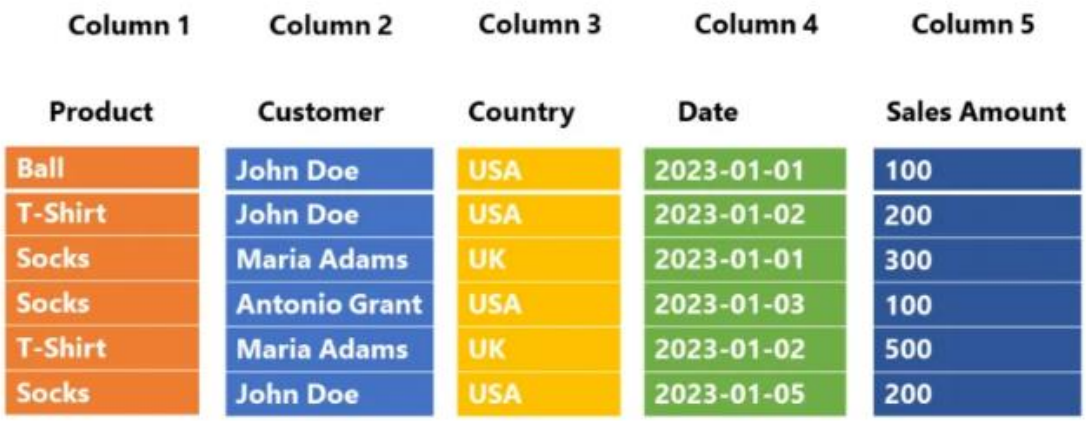
2.5 Αρχιτεκτονική Parquet αρχείου

Σε αυτό το σημείο καλό θα είναι να γίνει εμβάθυνση στις περίπλοκες λεπτομέρειες του τρόπου δομής των αρχείων Parquet, του τρόπου οργάνωσης των μεταδεδομένων και των σελίδων δεδομένων και πως

αυτού του είδους τα αρχεία έχουν κρίσιμο ρόλο στη βελτιστοποίηση της αποθήκευσης δεδομένων και της απόδοσης ερωτημάτων.

2.6 Δομή αρχείων Parquet

Τα αρχεία Parquet οργανώνονται σε μορφή στήλης αποθήκευσης (column store), πράγμα που σημαίνει ότι αντί να αποθηκεύει δεδομένα σε σειρές (CSV) όπως οι παραδοσιακές βάσεις δεδομένων, το Parquet αποθηκεύει δεδομένα σε στήλες.



Column 1	Column 2	Column 3	Column 4	Column 5
Product	Customer	Country	Date	Sales Amount
Ball	John Doe	USA	2023-01-01	100
T-Shirt	John Doe	USA	2023-01-02	200
Socks	Maria Adams	UK	2023-01-01	300
Socks	Antonio Grant	USA	2023-01-03	100
T-Shirt	Maria Adams	UK	2023-01-02	500
Socks	John Doe	USA	2023-01-05	200

- Στήλες και ομάδες γραμμών (Row-Groups):

Τα δεδομένα σε ένα αρχείο Parquet χωρίζονται σε οριζόντιες «φέτες» που ονομάζονται RowGroups. Κάθε ομάδα σειρών περιέχει ένα τμήμα των δεδομένων και οι στήλες μέσα σε μια ομάδα σειρών αποθηκεύονται μαζί για τη βελτιστοποίηση της συμπίεσης και την ελαχιστοποίηση των λειτουργιών εισόδου/εξόδου.

	Column 1	Column 2	Column 3	Column 4	Column 5
	Product	Customer	Country	Date	Sales Amount
Row Group 1	Ball	John Doe	USA	2023-01-01	100
	T-Shirt	John Doe	USA	2023-01-02	200
Row Group 2	Socks	Maria Adams	UK	2023-01-01	300
	Socks	Antonio Grant	USA	2023-01-03	100
Row Group 3	T-Shirt	Maria Adams	UK	2023-01-02	500
	Socks	John Doe	USA	2023-01-05	200

- Μεταδεδομένα (Metadata) και Σελίδες Δεδομένων (Data Pages) :

Τα αρχεία παρκέ περιέχουν μεταδεδομένα που περιγράφουν τη δομή των δεδομένων και επιτρέπουν την αποτελεσματική ανάκτηση τους. Υπάρχουν τρεις κύριοι τύποι μεταδεδομένων: μεταδεδομένα αρχείου, μεταδεδομένα στήλης(κομμάτι) και μεταδεδομένα κεφαλίδας σελίδας.

File Metadata: (Data about data) Πληροφορίες υψηλού επιπέδου σχετικά με το αρχείο Parquet, συμπεριλαμβανομένης της έκδοσης, του σχήματος, των ομάδων σειρών και του υποσέλιδου, που επιτρέπουν την αποτελεσματική πλοήγηση στα αρχεία και την ανάκτηση δεδομένων.

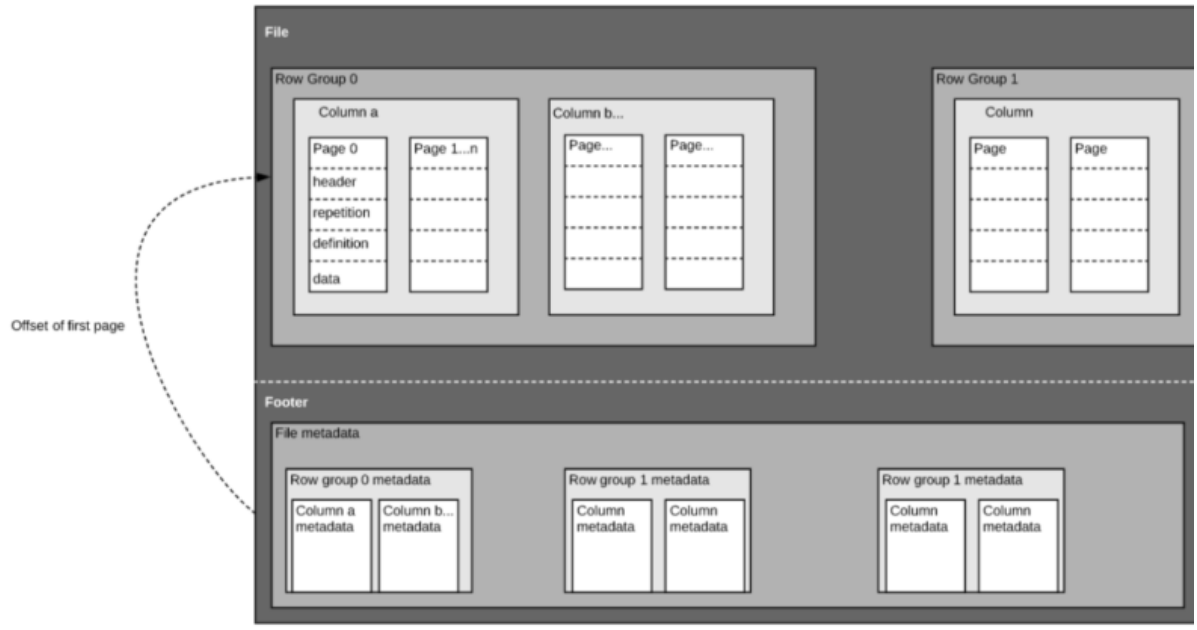
Column Chunk Metadata: Λεπτομέρειες για συγκεκριμένες στήλες μέσα σε μια ομάδα σειρών row groups.

Πιο αναλυτικά:

1) Πληροφορίες σχήματος του αρχείου, όπως ονόματα και τύποι στηλών
2) Οι θέσεις των RowGroup και ColumnChunks στο αρχείο

3) Στατιστικά στοιχεία για κάθε ColumnChunk, συμπεριλαμβανομένων των ελάχιστων/μέγιστων τιμών και των μηδενικών τιμών
4) Δείκτες σε OffsetIndexes που περιέχουν τη θέση κάθε μεμονωμένης σελίδας
5) Δείκτες στο ColumnIndex που περιέχουν μετρήσεις σειρών και συνοπτικά στατιστικά στοιχεία για κάθε σελίδα
6) Δείκτες στο BloomFilterData, οι οποίοι μπορούν γρήγορα να ελέγξουν εάν υπάρχει μια τιμή σε ένα ColumnChunk

Page Header Metadata: Πληροφορίες σε κάθε σελίδα δεδομένων, όπως μέγεθος, αναφορές λεξικού, κωδικοποίηση και μέτρηση τιμών, διευκολύνοντας την αποτελεσματική αποκωδικοποίηση και επεξεργασία στηλών δεδομένων κατά τη διάρκεια των ερωτημάτων.



Φωτογραφία 1

Τα αρχεία παρκέ αποτελούνται από ομάδες σειρών, κεφαλίδα και υποσέλιδο (Φωτογραφία 1). Κάθε ομάδα σειρών περιέχει δεδομένα από τις ίδιες στήλες. Οι ίδιες στήλες αποθηκεύονται μαζί σε κάθε ομάδα γραμμών.

- Κωδικοποίηση λεξικού (Dictonary Encoding):

Η κωδικοποίηση λεξικού είναι μια τεχνική που χρησιμοποιείται για τη συμπίεση επαναλαμβανόμενων ή περιττών δεδομένων. Σε μια μορφή στηλών αποθήκευσης όπως το Parquet, υπάρχει συχνά μεγάλη πιθανότητα επανάληψης δεδομένων σε μία στήλη. Η κωδικοποίηση λεξικού αντικαθιστά επαναλαμβανόμενες τιμές με μικρότερα αναγνωριστικά (ή δείκτες) που αναφέρονται σε ένα λεξικό μοναδικών τιμών.

Product	Index
Ball	0
T-Shirt	1
Socks	2
Socks	2
T-Shirt	1
Socks	2

Index	Product
0	Ball
1	T-Shirt
2	Socks

Πλεονεκτήματα: Αυτή η τεχνική αποθηκεύει τα επαναλαμβανόμενα δεδομένα μόνο μία φορά στο λεξικό. Επίσης, βελτιώνει την απόδοση του ερωτήματος, καθώς οι στήλες που κωδικοποιούνται από λεξικό μπορούν να συμπιεστούν και να αποσυμπιεστούν αποτελεσματικά κατά την εκτέλεση του ερωτήματος [όπως επισημαίνεται στην 4η βιβλιογραφία].

2.7 Πότε πρέπει να χρησιμοποιείται ο συγκεκριμένος τρόπος αποθήκευσης δεδομένων;

1) Όταν ο όγκος των δεδομένων είναι αρκετά μεγάλος. Το Parquet είναι κατασκευασμένο για απόδοση και αποτελεσματική συμπίεση.
2) Όταν το σύνολο δεδομένων έχει πολλές στήλες, αλλά χρειάζεται μόνο πρόσβαση σε ένα υποσύνολο. Λόγω της αυξανόμενης πολυπλοκότητας των επιχειρηματικών δεδομένων που καταγράφονται καθημερινός, μπορεί να διαπιστωθεί ότι αντί να συλλεχθούν 20 πεδία για κάθε συμβάν δεδομένων, καταγράφονται πλέον 100+. Ενώ αυτά τα δεδομένα αποθηκεύονται εύκολα σε μια ‘’λίμνη’’ δεδομένων (Data Lake), η αναζήτηση θα απαιτήσει σάρωση σημαντικού όγκου δεδομένων εάν αποθηκευτούν σε μορφές που βασίζονται σε γραμμές. Η στηλώδης και αυτοπεριγραφόμενη φύση του Parquet επιτρέπει να αντληθούν μόνο οι απαιτούμενες στήλες που απαιτούνται για συγκεκριμένα ερώτημα, μειώνοντας τον όγκο των δεδομένων που υποβάλλονται σε επεξεργασία.
3) Όταν πολλές υπηρεσίες να καταναλώνουν τα ίδια δεδομένα από την αποθήκευση αντικειμένων. Ενώ οι προμηθευτές βάσεων δεδομένων όπως η Oracle και η Snowflake προτιμούν να αποθηκεύουμε τα δεδομένα σε ιδιόκτητη μορφή που μόνο τα εργαλεία τους μπορούν να διαβάσουν, η σύγχρονη αρχιτεκτονική δεδομένων είναι προκατειλημμένη ως προς την αποσύνδεση της αποθήκευσης από τον υπολογιστή. Εάν κρίνεται η ανάγκη εργασίας με πολλές υπηρεσίες ανάλυσης των δεδομένων για να απαντηθούν σε διαφορετικές περιπτώσεις χρήσης, θα πρέπει τα δεδομένα να αποθηκευτούν σε Parquet files [όπως φαίνεται στην βιβλιογραφία 3].

Κεφάλαιο 3: Περιγραφή Κώδικα

3.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο γίνεται μια αναλυτική περιγραφή του κώδικα που έχει υλοποιηθεί, μέσα από τον οποίο έχουν γίνει τα πειράματα.

Πιο συγκεκριμένα υπάρχουν πληροφορίες για το πόσα και ποια αρχεία έχουν χρησιμοποιηθεί, τι είδους αρχεία είναι, τι επεξεργασία έχουν υποστεί, τι είδους ερωτήματα έχουν εφαρμοστεί σε αυτά, καθώς και πάνω σε ποιες μετρήσεις έχουν στηριχτεί οι δοκιμές και κατ' επέκταση τα αποτελέσματα που έχουν προκύψει.

3.2 Περιγραφή κώδικα

Ο κώδικας έχει ως σκοπό την εξαγωγή ορισμένων μετρήσεων των οποίων που έχουν εξαχθεί εφαρμόζοντας ορισμένα ερωτήματα πάνω στα CSV και Parquet αρχεία.

Σημείωση:

Το Sorted Parquet file δεν είναι απλά ταξινομημένο, όπως υποδηλώνει το όνομα του, αλλά έχει υποστεί και μία παραπάνω επεξεργασία ως προς την οριοθέτηση των γραμμών των Row Groups που θα περιέχει, καθώς είναι μια από τις τεχνικές βελτίωσης του.

Τα ερωτήματα είναι τα εξής:

- Select queries πάνω στα αρχεία (CSV, Parquet, Sorted Parquet) με φιλτράρισμα σε μία ή πολλές στήλες, με αριθμούς τυχαίους, συνθήκες στην περίπτωση της δημιουργίας τους από την μηχανή τυχαίους, εν αντιθέσει στην περίπτωση υλοποίησης των ερωτημάτων από τον χρήστη που επιλέγει τις συνθήκες εκείνος. (Στην περίπτωση των πολλών στηλών τα φίλτρα συνδέονται με το OR, για να την διευκόλυνση των πειραμάτων μας).

Οι μετρήσεις είναι οι εξής:

- Χρόνος που διαβάζει το κάθε αρχείο ξεχωριστά.
- Το μέγεθος του κάθε αρχείου.
- Χρόνος εκτέλεσης για τρία διαφορετικά queries (μετρώντας λίγα, μέτρια και πολλά αποτελέσματα.
- Μνήμη RAM για την οποία χρησιμοποιεί για κάθε query στο κάθε αρχείο ξεχωριστά.

Ο κώδικας έχει σχεδιαστεί για τη συγκριτική αξιολόγηση της απόδοσης ερωτημάτων και της χρήσης μνήμης των αρχείων CSV και Parquet με βάση κριτήρια επιλεγμένα από τον χρήστη ή τη μηχανή.

Είναι αρκετά διαδραστικός δίνοντας την δυνατότητα στον χρήστη να αλληλεπιδράσει με το πρόγραμμα, επιλέγοντας ο ίδιος των αριθμό των στηλών, τις στήλες μία προς μία, και φυσικά τα conditions για κάθε στήλη ξεχωριστά, ώστε να δημιουργεί μόνος του κατά αυτό τον τρόπο την σύνθεση των queries, μέχρι φυσικά εκείνος να σταματήσει.

Ωστόσο δίνεται και η δυνατότητα επιλογής του χρήστη, να κάνει το πρόγραμμα όλη την πιο πάνω διαδικασία, λαμβάνοντας μόνο τον αριθμό των επαναλήψεων (από τον χρήστη), που επιθυμεί να υλοποιήσει.

3.3 Περιγραφή των αρχείων του κώδικα

Στην υλοποίηση υπάρχουν συγκεκριμένα ένα CSV αρχείο και δύο Parquet αρχεία.

Το CSV αρχείο:

- Έχει αντληθεί από τον συγκεκριμένο ιστότοπο (https://zenodo.org/records/1167595)

Τα Parquet αρχεία:

- Ένα Parquet αρχείο το οποίο είναι η αυτούσια μετατροπή του CSV σε Parquet.
- Ένα ακόμα Parquet αρχείο το οποίο δημιουργείται ταξινομημένο με βάση την επιλογή της στήλης από τον χρήστη.

3.4 Ανάλυση κώδικα

3.4.1 Εισαγωγή

Ο κώδικας ξεκινάει εισάγοντας τις απαραίτητες βιβλιοθήκες όπως panda, time, psutil, os, matplotlib.pyplot, numpy και pyarrow, οι οποίες είναι απαραίτητες για την υλοποίηση.
Αρχικοποιεί λεξικά για να αποθηκεύει χρόνους ερωτημάτων και χρήση μνήμης RAM για αρχεία CSV, Parquet και Sorted Parquet, ώστε να μπορούν ύστερα να αποτυπωθούν στα διαγράμματα μας.

3.4.2 Διάβασμα αρχείων

- CSV: Αφού κατέβηκε το συγκεκριμένο αρχείο τοπικά στον υπολογιστή, ως «nari_dynamic.csv», το διαβάζεται με την χρήση της συνάρτησης read_csv() της βιβλιοθήκης pandas, ως pandas dataframe.

- PARQUET: Το Parquet αρχείο με όνομα «nari_dynamic_prq.parquet» χρειάστηκε αρχικά να δημιουργηθεί μέσα από το ήδη υπάρχων CSV file που έχουμε και με την βοήθεια της write_table() συνάρτησης της βιβλιοθήκης pyarrow, ως pyarrow table.

3.4.3 Τα πρώτα metrics

Πάνω σε αυτά λοιπόν τα αρχεία πραγματοποιούνται και τα πρώτα μας metrics:

- Χρόνος ανάγνωσης του κάθε αρχείου.
- Μέγεθος του κάθε αρχείου.

Γίνεται σαφέστατη και αντιληπτή η διαφορά μεταξύ των δύο αρχείων, πολύ μεγαλύτερο το size του CSV σε σχέση με το parquet, καθώς επίσης ανάλογη είναι και η διαφορά στον χρόνο ανάγνωσης των δύο αρχείων.

3.4.4 Επαναληπτική Δομή Κώδικα

Ύστερα από τα παραπάνω, ξεκινάει η επαναληπτική ροή – αλληλεπιδραστικό μέρος του κώδικα μας όπου ο χρήστης είναι εκείνος που θα διαλέξει με ποιον τρόπο επιθυμεί να λειτουργήσει ο κώδικας μας.

<u>Πιο συγκεκριμένα:</u>

Υπάρχουν δύο επιλογές για την λειτουργία του προγράμματος:

- Αλληλεπίδραση προγράμματος (όπου οι επιλογές για την ταξινόμηση και την δομή των queries είναι στην διακριτική ευχέρεια του προγράμματος καθαρά και μόνο)
- Αλληλεπίδραση χρήστη – προγράμματος (όπου οι επιλογές για την ταξινόμηση και την δομή των queries είναι στην διακριτική ευχέρεια του χρήστη καθαρά και μόνο)

Ωστόσο, δίνεται και η επιλογή να κάνει switch την αλληλεπίδραση κάθε φορά που τελειώνει το πρόγραμμα, δηλαδή μπορεί να επιθυμεί ο ίδιος να ξεκινήσει έχοντας την επιλογή εκείνος της δημιουργίας των queries, και μετά να επιθυμεί να την μεταβιβάσει την επιλογή στο πρόγραμμα δίνοντας απλά και μόνο τον αριθμό των επαναλήψεων.

3.4.5 Αλληλεπίδραση Προγράμματος

Ο χρήστης μεταβαίνει σε αυτή την λειτουργικότητα – αλληλεπίδραση εισάγοντας την λέξη «Machine», στο πρώτο ερώτημα που του παρατίθενται από το πρόγραμμα.

Έπειτα καλείται να δώσει και τον αριθμό των επαναλήψεων που επιθυμεί να τρέξει ο κώδικας μας και όλα τα άλλα γίνονται πλέον αυτόματα.

Τι γίνεται αυτόματα και για κάθε επανάληψη;

Αυτόματα το πρόγραμμα μας επιλέγει σε ποια στήλη αρχικά θέλει να ταξινομήσει το Parquet αρχείο μας,

όπου και αποθηκεύεται ως «sorted_nary_dynamic.parquet».

Στην συνέχεια επιλέγει σε ποια/ποιες στήλη/στήλες θα επιλέξει να δημιουργήσει το query του καθώς επίσης και την συνθήκη για κάθε στήλη ξεχωριστά αλλά και τον τυχαίο αριθμό για κάθε στήλη, ο οποίος βρίσκεται μεταξύ της min και την max τιμής κάθε στήλης.

Το query λοιπόν που έχει δημιουργήσει το πρόγραμμα μας θα εφαρμοστεί στα τρία μας αρχεία:

- nari_dynamic.csv
- nari_dynamic_prq.parquet
- sorted_nary_dynamic.parquet

ώστε κατά την διεξαγωγή τους να αντλήσουμε τα metrics:

- χρόνος εκτέλεσης
- μνήμη RAM που χρησιμοποιήθηκε

και να τα εναποθέσουμε στα κατάλληλα dictionaries.

3.4.6 Αλληλεπίδραση χρήστη – προγράμματος

Η ίδια ακριβώς διαδικασία ακολουθείται και εδώ, με την διαφορά ότι το πρόγραμμα μας ζητάει συνεχώς inputs όπως:

- Την στήλη στην οποία επιθυμεί να ταξινομήσει το Parquet αρχείο μας
- Τον αριθμό των στηλών που επιθυμεί να βάλει στο query
- Μία προς μία τις στήλες που επιθυμεί συγκεκριμένα
- Τις συνθήκες για κάθε στήλη ξεχωριστά

από τον χρήστη ώστε να δημιουργηθούν τα queries μας.

Οι αριθμοί για κάθε στήλη και εδώ επιλέγονται από το πρόγραμμα με τον ίδιο ακριβώς τυχαίο τρόπο όπως εξηγήσαμε πιο πάνω, με βάση μέσα στο διάστημα min – max τιμής κάθε στήλης.

3.4.7 Sorted Parquet File

Σε αυτό το σημείο καλό θα ήταν να γίνει αναφορά στο συγκεκριμένο αρχείο, ώστε να γίνει κατανοητό τι επεξεργασία έχει υποστεί και για πιο λόγο.

Όπως έχει αναφερθεί και πιο πάνω το αρχείο sorted_nary_dynamic_prq.parquet, έχει δημιουργηθεί με βάση το αρχείο μας nary_dynamic_prq.parquet, αλλά με ορισμένες επεξεργασίες.

Ποιες είναι αυτές;

- Ταξινόμηση: Ταξινόμηση και συγκεκριμένα σε φθίνουσα σειρά (κάθε φορά με βάση της στήλης που επιθυμεί ο χρήστης ή το πρόγραμμα μας), όπως γίνεται αντιληπτό η ταξινόμηση δεν είναι τίποτα παραπάνω από τον τρόπο που οργανώνονται τα rows μας (οι γραμμές μας) με βάση την ταξινόμηση μίας η παραπάνω γραμμής σε φθίνουσα ή αύξουσα σειρά.

- Row groups: Καλό θα ήταν σε αυτό το σημείο να σταθούμε παραπάνω για να γίνει πλήρως κατανοητή η έννοια των row groups. Τα row groups είναι ένας ιδιαίτερος και ειδικός τρόπος οργάνωσης της πληροφορίας, δηλαδή των δεδομένων που βρίσκονται στα Parquet αρχεία. Ουσιαστικά τα row groups είναι μια μορφή αποθήκευσης στηλών, που σημαίνει ότι τα δεδομένα αποθηκεύονται και οργανώνονται κατά στήλες και όχι σε γραμμές. Τα row groups είναι ένας τρόπος ομαδοποίησης ενός συγκεκριμένου αριθμού σειρών. Κάθε ομάδα γραμμών σε ένα αρχείο Parquet περιέχει δεδομένα για όλες τις στήλες (τα column chunks που δημιουργούνται αποθηκεύονται μέσα σε data pages), αλλά μόνο για ένα υποσύνολο των γραμμών. Η επεξεργασία λοιπόν που έχει υποστεί το αρχείο μας είναι ο καθορισμός των row groups, δηλαδή έχουμε ορίσει εμείς οι ίδιοι στο αρχείο μας πόσες γραμμές θα αποθευόνται.

Σε αυτό το σημείο θα γίνει περεταίρω εμβάθυνση, με την βοήθεια μερικών εικόνων από τα ενδώτερα των δύο Parquet αρχείων, ώστε να γίνει πιο αντιληπτή η έννοια των row groups και ο λόγος της συγκεκριμένης επεξεργασίας.

1) nari_dynamic_prq.parquet: Σε αυτό το αρχείο που δεν έχει υποστεί κάποια επεξεργασία παρατηρούμε ότι ο αριθμός των row groups είναι μόλις 19 στο σύνολο για ένα dataset το οποίο έχει εκατομμύρια γραμμές που σημαίνει ότι όλη η πληροφορία βρίσκεται διασπασμένη μέσα σε

αυτά τα 19 row groups. Παρατηρείται ότι κάθε row group περιέχει όλες τις στήλες του dataset μας, καθώς και συγκεκριμένο αριθμό γραμμών, όπως απεικονίζεται.

```
Number of Row Groups: 19
Row Group: 0
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 23.63 MB

Row Group: 1
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 22.78 MB

Row Group: 2
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 22.99 MB

Row Group: 3
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 23.99 MB

Row Group: 4
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 22.53 MB

Row Group: 5
Number of Rows: 1048576
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't']
Total Byte Size: 23.76 MB
```

2) sorted_dynamic_prq.parquet: Το συγκεκριμένο Parquet αρχείο όπως φαίνεται και στην παρακάτω φωτογραφία, αποτελείται από 1904 row groups (έναντι 19 του προηγούμενου), που σημαίνει πως η πληροφορία μας έχει κατανεμηθεί μέσα σε αυτά τα row groups (όντας και ταξινομημένα μέσα σε αυτά), με κάθε data page πολύ μικρότερο σε μέγεθος σε σχέση με το προηγούμενο μας αρχείο, που είναι και απόλυτα λογικό, καθώς πριν σε είχαμε για σε κάθε row group αριθμό γραμμών της τάξεως του εκατομμυρίου, εν αντιθέσει με αυτό το αρχείο που του έχουμε ορίσει τον αριθμό για κάθε row group στα 10.000. (Το «__index_level_0__» δεν αποτελεί στήλη του πίνακα, αλλά μια metadata πληροφορία για το index της κάθε γραμμής)

```
Number of Row Groups: 1904
Row Group: 0
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.35 MB

Row Group: 1
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.35 MB

Row Group: 2
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.35 MB

Row Group: 3
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.35 MB

Row Group: 4
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.33 MB

Row Group: 5
Number of Rows: 10000
Column Names: ['sourcemmsi', 'navigationalstatus', 'rateofturn', 'speedoverground', 'courseoverground', 'trueheading', 'lon', 'lat', 't', '__index_level_0__']
Total Byte Size: 0.35 MB
```

(Οι παραπάνω απεικονίσεις, δεν αποτελούν μέρος του κώδικα, επισυνάπτονται για να γίνει αντιληπτός ο λόγος της συγκεκριμένης επεξεργασίας, αλλά και ο τρόπος οργάνωσης την πληροφορίας μέσα στα Parquet files.)

Τι μπορεί να σημαίνουν όμως τα παραπάνω μεταξύ των δύο Parquet αρχείων;

Τα παραπάνω μπορεί να σημαίνουν σημαντική βελτίωση και αποδοτικότερη συμπεριφορά του sorted αρχείου, σε σχέση με το αρχικό Parquet αρχείο και αυτό γιατί όχι μόνο η πληροφορία είναι ταξινομημένη, αλλά και γιατί πλέον τα row groups είναι μικρότερα σε μέγεθος που σημαίνει ότι η υπολογιστική μηχανή σηκώνει πλέον μικρότερα σε μέγεθος αρχεία και σε συνδυασμό με την ταξινόμηση και τα metadata που υπάρχουν σε κάθε data page η απάντηση στα ερωτήματα γίνεται πιο γρήγορη.

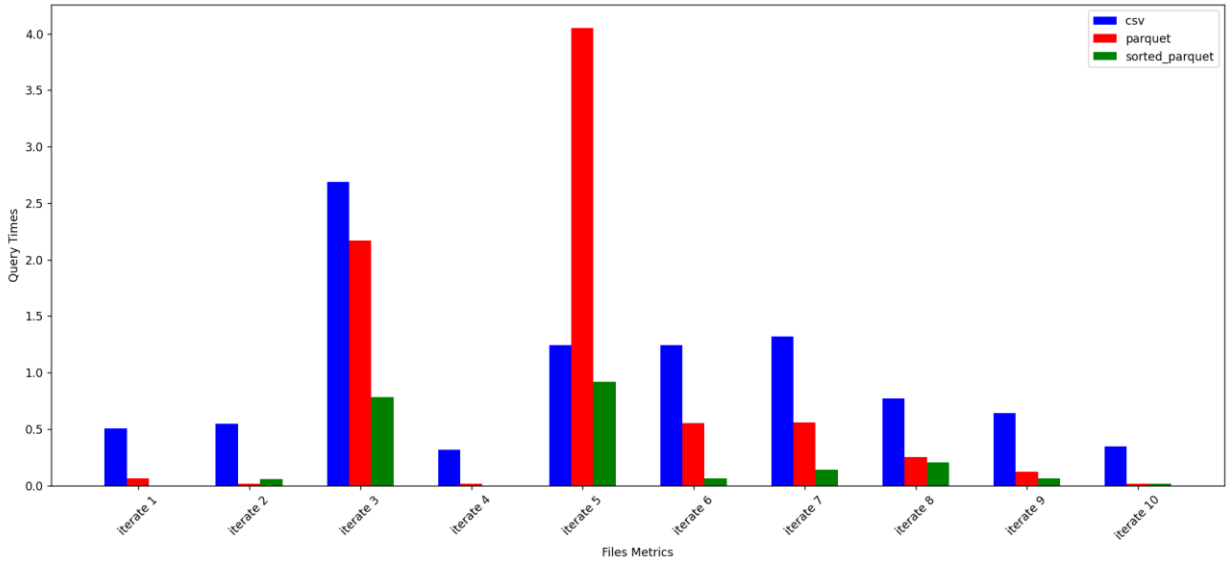
Κεφάλαιο 4: Αποτελέσματα-Διαγράμματα

4.1 Γενικά αποτελέσματα

```
***** CSV FILE INFO *****
The length of all csv table is: 19035630 rows
Time to read the CSV: 9.56547 sec
The CSV size is: 1.0473 GB
***** PARQUET FILE INFO *****
The length of all parquet table is: 19035630 rows
Time to read the PARQUET: 0.49047 sec
The PARQUET size is: 0.29042 GB
```

Στην αρχή της εξαγωγής των μετρήσεων ως προς τα γενικά μετρήσιμα, φαίνονται ξεκάθαρα οι διαφορές σε πρώτο επίπεδο του CSV-PARQUE ως προς το μέγεθος και τον χρόνο ανάγνωσης από την μηχανή.

4.2 Διάγραμμα χρόνου εκτέλεσης των ερωτημάτων



Το παραπάνω διάγραμμα μας απεικονίζει τον χρόνο εκτέλεσης που απαιτήθηκε για κάθε αρχείο στο να απαντήσει στο κάθε ερώτημα που του έχει ανατεθεί.

Με μια γρήγορη ματιά γίνεται αντιληπτό ότι πράγματι τα πειράματα, επιστρέφουν τα αποτελέσματα τα οποία ήταν αναμενόμενα, δηλαδή η απόδοση του επεξεργασμένου Parquet αρχείου να υπερτερεί έναντι του αρχικού Parquet αρχείου και το αρχικό Parquet αρχείο υπερτερεί σε απόδοση από το CSV. Παρόλο αυτά η απόδοση του CSV μπορεί να είναι πιο αποδοτική σε μερικές περιπτώσεις σε σχέση με τα υπόλοιπα δύο αρχεία Parquet.

Ας δούμε όμως εκ των έσω κάθε επανάληψη ξεχωριστά μαζί τι αποτελέσματα μας έχει επιστρέψει.

Επανάληψη 1:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.0518 sec
The SORTED PARQUET size is: 0.43938 GB
Enter the number of columns you want to query: 1
Enter the column number 1: 1
Enter the condition for column 'sourcemmsi': >
{'sourcemmsi': '>'}
{'sourcemmsi': 939224040}
The length of the original csv 19035630 rows
CSV LENGTH 370087 rows
CSV TIME 0.50449 sec
CSV MEMORY 0.53966 GB
The length of the original parquet 19035630
PARQUET LENGHT 370087 rows
PARQUET TIME 0.06247 sec
PARQUET MEMORY 0.13029 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 370087 rows
SORTED PARQUET TIME 0.0 sec
SORTED PARQUET MEMORY 0.00977 GB
```

Σε αυτή την επανάληψη το query το οποίο μπορεί να μην δείχνει τόσο περίπλοκο καθώς εκτελείται πάνω σε μία στήλη μόνο, παρόλο αυτά μας επιστρέφει πολύ λίγες γραμμές σε σχέση με το αρχικό πολύ μεγάλο σύνολο δεδομένων που είχαμε.

Τι σημαίνει αυτό;

Σημαίνει ότι το CSV αρχείο για ερωτήματα που επιστρέφουν λίγες εγγραφές, που χρειάζεται δηλαδή να διαβάσει όλο το αρχείο και να διαπεράσει όλη την πληροφορία, δυσκολεύεται παραπάνω σε σχέση με τα Parquet αρχεία, και αυτό γιατί αναγκάζεται να διαβάσει όλο το αρχείο σε σχέση με το Parquet που έχοντας οργανωμένη μέσα την πληροφορία του, σε συνδυασμό και με τα meta-data που διαθέτει ανταποκρίνεται καλύτερα σε τέτοιου είδους ερωτήματα διαβάζοντας μόνο αυτά που χρειάζεται.

Επανάληψη 2:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 0.61167 sec
The SORTED PARQUET size is: 0.41585 GB
Enter the number of columns you want to query: 3
Enter the column number 1: 1
Enter the column number 2: 2
Enter the column number 3: 3
Enter the condition for column 'sourcemmsi': >
Enter the condition for column 'navigationalstatus': <=
Enter the condition for column 'rateofturn': >
{'sourcemmsi': '>', 'navigationalstatus': '<=', 'rateofturn': '>'}
{'sourcemmsi': 692491515, 'navigationalstatus': 4, 'rateofturn': -28}
The length of the original csv 19035630 rows
CSV LENGTH 10013 rows
CSV TIME 0.54945 sec
CSV MEMORY 0.21727 GB
The length of the original parquet 19035630
PARQUET LENGHT 10013 rows
PARQUET TIME 0.01563 sec
PARQUET MEMORY 0.00092 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 10013 rows
SORTED PARQUET TIME 0.06027 sec
SORTED PARQUET MEMORY 0.00861 GB
```

Σε αυτή την περίπτωση παρατηρείται ένα πιο περίπλοκο query το οποίο επιστρέφει επίσης λίγες εγγραφές, με τα Parquet αρχεία να είναι και πιο αποδοτικά, όπως και στην πρώτη επανάληψη. Σε αυτή την επανάληψη διακρίνεται το εξής, το αρχικό Parquet αρχείο να είναι πιο αποδοτικό σε σχέση με το Sorted Parquet αρχείο, πράγμα που σημαίνει ότι για το συγκεκριμένο query θα έπρεπε να αυξηθεί το μέγεθος των Row Groups, γιατί χρειάστηκε να διαπεράσει η μηχανή αρκετά από αυτά, με αποτέλεσμα να

καθυστερήσει να απαντήσει στο query. Άρα θα έπρεπε εδώ να του αυξηθεί ο αριθμός των Row Groups του.

Επανάληψη 3:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.13181 sec
The SORTED PARQUET size is: 0.46873 GB
Enter the number of columns you want to query: 2
Enter the column number 1: 5
Enter the column number 2: 6
Enter the condition for column 'courseoverground': >
Enter the condition for column 'trueheading': >=
{'courseoverground': '>', 'trueheading': '>='}
{'courseoverground': 283, 'trueheading': 389}
The length of the original csv 19035630 rows
CSV LENGTH 3247268 rows
CSV TIME 2.68711 sec
CSV MEMORY 1.02299 GB
The length of the original parquet 19035630
PARQUET LENGHT 3247268 rows
PARQUET TIME 2.16642 sec
PARQUET MEMORY 0.64901 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 3247268 rows
SORTED PARQUET TIME 0.78507 sec
SORTED PARQUET MEMORY 0.17771 GB
```

Εδώ επίσης έχουμε μία ακριβώς παρόμοια περίπτωση με αυτή της πρώτης επανάληψης.

Επανάληψη 4:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.7035 sec
The SORTED PARQUET size is: 0.47114 GB
Enter the number of columns you want to query: 4
Enter the column number 1: 1
Enter the column number 2: 2
Enter the column number 3: 3
Enter the column number 4: 4
Enter the condition for column 'sourcemmsi': >
Enter the condition for column 'navigationalstatus': <
Enter the condition for column 'rateofturn': <
Enter the condition for column 'speedoverground': >=
{'sourcemmsi': '>', 'navigationalstatus': '<', 'rateofturn': '<', 'speedoverground': '>='}
{'sourcemmsi': 515327372, 'navigationalstatus': 3, 'rateofturn': 105, 'speedoverground': 71}
The length of the original csv 19035630 rows
CSV LENGTH 303 rows
CSV TIME 0.315 sec
CSV MEMORY 0.232 GB
The length of the original parquet 19035630
PARQUET LENGHT 303 rows
PARQUET TIME 0.01562 sec
PARQUET MEMORY 0.00049 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 303 rows
SORTED PARQUET TIME 0.0 sec
SORTED PARQUET MEMORY 0.00334 GB
```

Και εδώ επίσης ακόμα μια επανάληψη με πολλές στήλες στο φίλτρο μας, επιστρέφοντας εξαιρετικά λίγες γραμμές αν ληφθεί υπόψη το αρχικό μέγεθος του αρχείου, βλέποντας πόσο αποδοτικά μπορούν να απαντήσουν τα Parquet αρχεία σε τόσο περίπλοκα queries.

Επανάληψη 5:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.04672 sec
The SORTED PARQUET size is: 0.41075 GB
Enter the number of columns you want to query: 1
Enter the column number 1: 3
Enter the condition for column 'rateofturn': <
{'rateofturn': '<'}
{'rateofturn': -66}
The length of the original csv 19035630 rows
CSV LENGTH 9618461 rows
CSV TIME 1.24506 sec
CSV MEMORY 1.66015 GB
The length of the original parquet 19035630
PARQUET LENGHT 9618461 rows
PARQUET TIME 4.05236 sec
PARQUET MEMORY 0.31667 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 9618461 rows
SORTED PARQUET TIME 0.91835 sec
SORTED PARQUET MEMORY 0.02935 GB
```

Σε αυτή την επανάληψη παρατηρείται κάτι το οποίο ενδεχομένως να μην ήταν αναμενόμενο. Το CSV αρχείο να απαντάει πιο γρήγορα το ερώτημα σε σχέση με το Parquet.

Τι παρατηρείται με μία πρώτη ματιά;

Το ερώτημα επιστρέφει αρκετές εγγραφές που σημαίνει ότι, η μηχανή έχει διατρέξει ένα πολύ μεγάλο μέρος των αρχείων. Σε αυτή την περίπτωση όντως το CSV αρχείο θα είναι πιο αποδοτικό σε σχέση με το Parquet αρχείο (άλλωστε τα Parquet αρχεία έχουν σχεδιαστεί για να μπορούν να ανταποκρίνονται για επίπονα και δύσκολα queries), γιατί το CSV είναι ένα μη συμπιεσμένο αρχείο, το οποίο απλά η μηχανή το διαβάζει για να μας γυρίσει τα αποτελέσματα μας. Αυτή η απλότητα καθιστά πολύ εύκολη την ανάγνωση του, ειδικά για διαδοχικές σαρώσεις όπως σε αυτά τα δύο συγκεκριμένα πειράματα. Στο Parquet τα πράγματα δεν είναι έτσι. Το Parquet είναι ένα συμπιεσμένο αρχείο το οποίο περιέχει, το οποίο η μηχανή μας αρχικά πρέπει να το αποσυμπιέσει και έπειτα να διατρέξει και να διαβάσει τα εσωτερικά data pages τα οποία όταν το query μας θα μας φέρει πολλά αποτελέσματα τότε η μηχανή καλείται να διατρέξει πάρα πολλά εξ αυτών.

Επανάληψη 6:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.1885 sec
The SORTED PARQUET size is: 0.41585 GB
Enter the number of columns you want to query: 1
Enter the column number 1: 8
Enter the condition for column 'lat': <=
{'lat': '<='}
{'lat': 48}
The length of the original csv 19035630 rows
CSV LENGTH 715138 rows
CSV TIME 1.24187 sec
CSV MEMORY 1.10403 GB
The length of the original parquet 19035630
PARQUET LENGHT 715138 rows
PARQUET TIME 0.55334 sec
PARQUET MEMORY 0.11949 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 715138 rows
SORTED PARQUET TIME 0.06272 sec
SORTED PARQUET MEMORY 0.02638 GB
```

Η επανάληψη εδώ είναι αντίστοιχη με την επανάληψη 1.

Επανάληψη 7:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 0.96717 sec
The SORTED PARQUET size is: 0.46638 GB
Enter the number of columns you want to query: 5
Enter the column number 1: 1
Enter the column number 2: 2
Enter the column number 3: 3
Enter the column number 4: 4
Enter the column number 5: 5
Enter the condition for column 'sourcemmsi': <
Enter the condition for column 'navigationalstatus': >
Enter the condition for column 'rateofturn': <
Enter the condition for column 'speedoverground': <=
Enter the condition for column 'courseoverground': >=
{'sourcemmsi': '<', 'navigationalstatus': '>', 'rateofturn': '<', 'speedoverground': '<=', 'courseoverground': '>='}
{'sourcemmsi': 926795719, 'navigationalstatus': 13, 'rateofturn': 46, 'speedoverground': 99, 'courseoverground': 355}
The length of the original csv 19035630 rows
CSV LENGTH 93579 rows
CSV TIME 1.31943 sec
CSV MEMORY 0.6328 GB
The length of the original parquet 19035630
PARQUET LENGHT 93579 rows
PARQUET TIME 0.56184 sec
PARQUET MEMORY 0.21951 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 93579 rows
SORTED PARQUET TIME 0.14279 sec
SORTED PARQUET MEMORY 0.03646 GB
```

Και εδώ συμβαίνει ένα περίπλοκο επίσης query με πολλές στήλες, επιστρέφοντας λίγες εγγραφές, με τις αναμενόμενα αποτελέσματα μετρήσεων.

Επανάληψη 8:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.06013 sec
The SORTED PARQUET size is: 0.41075 GB
Enter the number of columns you want to query: 3
Enter the column number 1: 4
Enter the column number 2: 5
Enter the column number 3: 6
Enter the condition for column 'speedoverground': >
Enter the condition for column 'courseoverground': >=
Enter the condition for column 'trueheading': >
{'speedoverground': '>', 'courseoverground': '>=', 'trueheading': '>'}
{'speedoverground': 46, 'courseoverground': 88, 'trueheading': 316}
The length of the original csv 19035630 rows
CSV LENGTH 921186 rows
CSV TIME 0.76978 sec
CSV MEMORY 0.90606 GB
The length of the original parquet 19035630
PARQUET LENGHT 921186 rows
PARQUET TIME 0.25123 sec
PARQUET MEMORY 0.16579 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 921186 rows
SORTED PARQUET TIME 0.20426 sec
SORTED PARQUET MEMORY 0.07243 GB
```

Ίδια περίπτωση με τίς προηγούμενες.

Επανάληψη 9:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 1.06326 sec
The SORTED PARQUET size is: 0.47114 GB
Enter the number of columns you want to query: 2
Enter the column number 1: 7
Enter the column number 2: 8
Enter the condition for column 'lon': >=
Enter the condition for column 'lat': >=
{'lon': '>=', 'lat': '>='}
{'lon': -4, 'lat': 47}
The length of the original csv 19035630 rows
CSV LENGTH 51924 rows
CSV TIME 0.64377 sec
CSV MEMORY 0.32211 GB
The length of the original parquet 19035630
PARQUET LENGHT 51924 rows
PARQUET TIME 0.12552 sec
PARQUET MEMORY 0.05647 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 51924 rows
SORTED PARQUET TIME 0.06249 sec
SORTED PARQUET MEMORY 0.00249 GB
```

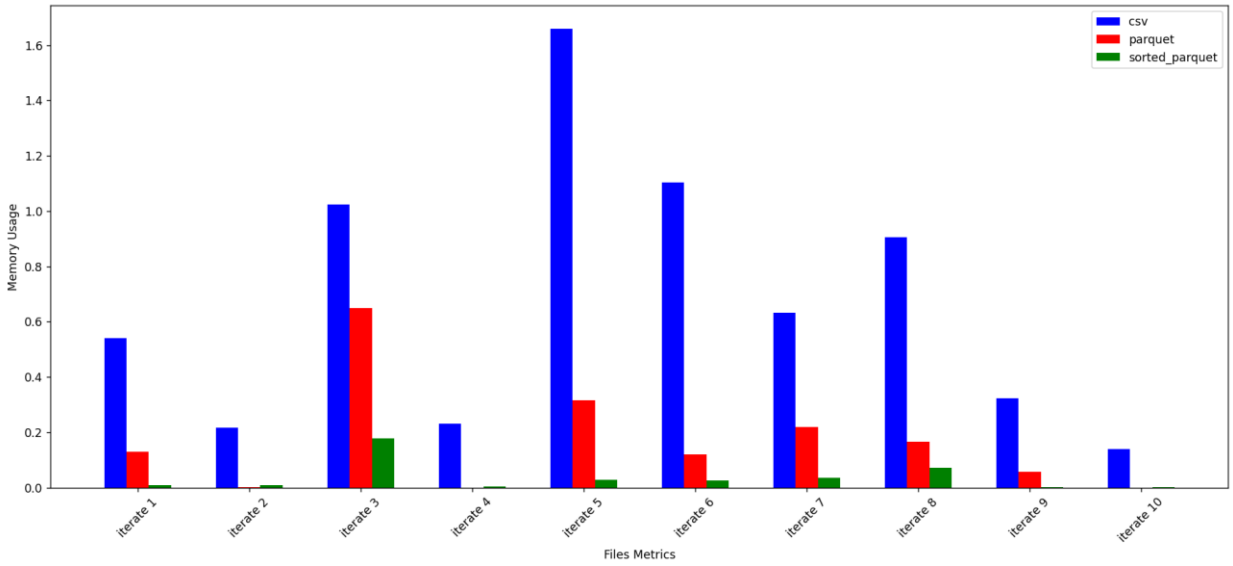
Και εδώ έχουμε τα αναμενόμενα αποτελέσματα.

Επανάληψη 10:

```
The length of all parquet table is: 19035630 rows
Time to read the SORTED PARQUET: 0.90458 sec
The SORTED PARQUET size is: 0.47165 GB
Enter the number of columns you want to query: 3
Enter the column number 1: 2
Enter the column number 2: 7
Enter the column number 3: 8
Enter the condition for column 'navigationalstatus': >
Enter the condition for column 'lon': >
Enter the condition for column 'lat': <=
{'navigationalstatus': '>', 'lon': '>', 'lat': '<='}
{'navigationalstatus': 9, 'lon': -4, 'lat': 51}
The length of the original csv 19035630 rows
CSV LENGTH 22402 rows
CSV TIME 0.34468 sec
CSV MEMORY 0.13965 GB
The length of the original parquet 19035630
PARQUET LENGHT 22402 rows
PARQUET TIME 0.01559 sec
PARQUET MEMORY 8e-05 GB
The length of the original sorted parquet 19035630
SORTED PARQUET LENGHT 22402 rows
SORTED PARQUET TIME 0.01562 sec
SORTED PARQUET MEMORY 0.00085 GB
```

Και σε αυτή την επανάληψη παρατηρούμε τα αναμενόμενα.

4.3 Διάγραμμα χρησιμοποίησης μνήμης από τα ερωτήματα



Το παραπάνω διάγραμμα απεικονίζει την μνήμη RAM που χρησιμοποιήθηκε για κάθε αρχείο σε κάθε ερώτημα.

Εμφανής η υπεροχή των Parquet files έναντι του CSV, καθώς πρόκειται για συμπιεσμένα αρχεία, πολύ μικρότερα σε μέγεθος σε σχέση με το CSV , άρα η μνήμη που απαιτείται από την μηχανή για να τα «σηκώσει» αλλά και για να τα διαβάσει είναι αρκετά μικρότερη.

Πιο αναλυτικά, για να υπολογιστεί ένα ερώτημα από CSV αρχείο το ολόκληρο το CSV αρχείο θα πρέπει να είναι «σηκωμένο» στην μνήμη για όση ώρα υπολογίζεται το ερώτημα, σε αντίθεση με τα Parquet αρχεία που κάθε φορά στην μνήμη απαιτείται να «σηκωθούν» μόνο τα data pages που περιέχουν την πληροφορία που θέλουμε, που από ότι είδαμε και από το εσωτερικό των Parquet αρχείων είναι πολύ μικρά σε μέγεθος, άρα και απαιτείται πολύ μικρότερη μνήμη RAM για να ανταποκριθεί στις προσδοκίες.

4.4 Συμπέρασμα

Το ταξινομημένο αρχείο Parquet βελτιώνει σημαντικά την απόδοση ερωτημάτων σε σύγκριση τόσο με το αρχικό αρχείο CSV όσο και με το μη ταξινομημένο αρχείο Parquet.

Η ταξινόμηση του αρχείου Parquet επιτρέπει πιο αποτελεσματική αναζήτηση, καθώς τα δεδομένα είναι οργανωμένα με τρόπο που ευθυγραμμίζεται με τις συνθήκες του ερωτήματος.

Ο μειωμένος χρόνος ερωτήματος και η χρήση μνήμης στο ταξινομημένο αρχείο Parquet υποδεικνύουν τα πλεονεκτήματα της προ-ταξινόμησης δεδομένων για ορισμένους τύπους ερωτημάτων, ιδιαίτερα εκείνων που αφορούν συνθήκες βάσει εύρους σε ταξινομημένες στήλες.

Συνολικά, αυτό δείχνει τα πλεονεκτήματα της αξιοποίησης των ταξινομημένων αρχείων Parquet για ορισμένα μοτίβα ερωτημάτων, συνήθως τα πιο περίπλοκα και απαιτητικά (άλλωστε και ο σκοπός που εξυπηρετούν τα Parquet files είναι αυτός ακριβώς να μπορέσει κάποια μορφή αρχείου να ανταποκρίνεται αποτελεσματικά σε περίπλοκα ερωτήματα), καθώς μπορούν να οδηγήσουν σε σημαντικές βελτιώσεις απόδοσης σε σύγκριση με μη ταξινομημένα δεδομένα ή άλλες μορφές αρχείων όπως το CSV.

Κεφάλαιο 5: Delta Tables

5.1 Εισαγωγή

Στην εποχή των μεγάλων δεδομένων, οι οργανισμοί αντιμετωπίζουν την πρόκληση της αποτελεσματικής διαχείρισης και επεξεργασίας τεράστιου όγκου δεδομένων, διασφαλίζοντας ταυτόχρονα όμως την ακεραιότητα, την αξιοπιστία και την επεκτασιμότητα των δεδομένων. Οι παραδοσιακές λύσεις αποθήκευσης δεδομένων συχνά δυσκολεύονται να ανταποκριθούν σε αυτές τις απαιτήσεις, οδηγώντας σε ζητήματα όπως ασυνέπεια δεδομένων, έλλειψη υποστήριξης συναλλαγών και δυσκολία στη διαχείριση των δεδομένων.

Τα Delta Tables, εμφανίστηκαν ως μια ισχυρή λύση για την αντιμετώπιση αυτών των προκλήσεων. Αποτελούν μια σύγχρονη και ισχυρή τεχνολογία που αντιμετωπίζει αποτελεσματικά τις προκλήσεις της διαχείρισης μεγάλων όγκων δεδομένων. Με χαρακτηριστικά όπως η υποστήριξη συναλλαγών ACID, ο επεκτάσιμος χειρισμός μεταδεδομένων και οι δυνατότητες εκδόσεων δεδομένων (versioning) για κάθε πίνακα, επιτρέπουν στους οργανισμούς να διαχειρίζονται δεδομένα με υψηλή απόδοση και αξιοπιστία.

5.2 Ορισμός Delta Tables

Οι Delta Tables είναι μια καλύτερη πιο εμπλουτισμένη έκδοση ενός κανονικού Table που προσφέρει επιπλέον λειτουργίες για την επεξεργασία των δεδομένα μας. Οι πίνακες Delta έχουν σχεδιαστεί για να παρέχουν έναν ισχυρό και αποτελεσματικό τρόπο διαχείρισης μεγάλων συνόλων δεδομένων σε λίμνες δεδομένων (Delta Lakes), ενώ προσφέρουν επίσης προηγμένες δυνατότητες που θα αναλύσουμε παρακάτω σε αντίθεση με τις παραδοσιακές μορφές δεδομένων.

Στον πυρήνα του πρακτικά, ένα Delta Table είναι ένα σύνολο αρχείων Parquet, και metadata (delta log) που είναι αποθηκευμένα σε μια λίμνη δεδομένων, με ένα ειδικό αρχείο καταγραφής (metadata) συναλλαγών που παρακολουθεί όλες τις αλλαγές που έγιναν στα δεδομένα [σύμφωνα με την 9η και 11η βιβλιογραφία].

5.2.1 Τι είναι το Delta Log;

Το Delta Log είναι ένας φάκελος με αρχεία καταγραφής συναλλαγών που συμβαίνουν σε κάθε Delta Table και αποθηκεύεται στην ίδια διεύθυνση (path) που βρίσκονται και τα Parquet αρχεία κάθε πίνακα μέσα στο Delta Lake.

5.2.2 Τι είναι το Delta Lake;

Αρχικά τι είναι το Data Lake;

Data Lake ή λίμνη δεδομένων είναι ένας κεντρικός χώρος αποθήκευσης για δομημένα, ημιδομημένα και μη δομημένα ακατέργαστα δεδομένα. Το πρόβλημα με τα παραδοσιακά Data Lakes είναι ότι συχνά δεν

διαθέτουν ισχυρές δυνατότητες διαχείρισης δεδομένων, γεγονός που οδηγεί σε ζητήματα όπως η ποιότητα, η αξιοπιστία και η ακεραιότητα των δεδομένων. Τα παραπάνω ζητήματα λύνονται με το Delta Lake. Το Delta Lake είναι χτισμένο πάνω στο Apache Spark* και είναι ένα στρώμα αποθήκευσης δεδομένων ανοιχτού κώδικα που προσθέτει ορισμένες σοβαρές αναβαθμίσεις στις λίμνες δεδομένων, όπως οι συναλλαγές ACID, ο επεκτάσιμος χειρισμός μεταδεδομένων και δυνατότητα χειρισμού τόσο ροής όσο και ομαδικής επεξεργασίας δεδομένων [όπως περιγράφεται στην βιβλιογραφία 11].

* Apache Spark: Το Apache Spark είναι ένα ανοιχτού κώδικα κατανεμημένο σύστημα επεξεργασίας που χρησιμοποιείται για φόρτους εργασίας μεγάλων δεδομένων. Χρησιμοποιεί προσωρινή αποθήκευση στη μνήμη και βελτιστοποιημένη εκτέλεση ερωτημάτων για γρήγορα αναλυτικά ερωτήματα έναντι δεδομένων οποιουδήποτε μεγέθους.

5.3 Πλεονεκτήματα Delta Tables

1) ACID Transaction: Η υποστήριξη συναλλαγών ACID των Delta Tables τους καθιστά ιδανικούς για την αποθήκευση και την επεξεργασία δεδομένων.

2) Data Versioning: Η δυνατότητα παρουσίας των πολλών εκδόσεων σε κάθε πίνακα Delta είναι επίσης πολύ χρήσιμη για σκοπούς εντοπισμού σφαλμάτων, ελέγχου των δεδομένων και συμμόρφωσης τους. Όλες οι αλλαγές και οι παραμετροποιήσεις που υπόκεινται είναι εμφανής, πράγμα που βοηθάει στην αποσφαλμάτωση, και στην επαναφορά της σωστής κατάστασης του πίνακα σε περίπτωση βλάβης.

3) Time Travel: Η δυνατότητα μετάβασης σε παρελθοντικές καταστάσεις (states) του πίνακα, είναι ένα άλλο ισχυρό εργαλείο που μπορεί να χρησιμοποιηθεί για ανάλυση δεδομένων. Η αναζήτηση σε με λίγα λόγια σε ιστορικά δεδομένα σαν να είναι ακόμα παρόντα.

4) Scalability: Το Delta Lake έχει σχεδιαστεί για κλίμακα σε petabyte και exabyte δεδομένων, καθιστώντας το ιδανικό για αποθήκευση και επεξεργασία μεγάλων συνόλων δεδομένων, για την διαχείριση πολλών αρχείων και metadata [όπως αναγράφεται στην βιβλιογραφία 11].

5.4 Κατηγορίες Delta Tables

- Managed Delta Tables: Είναι οι πίνακες όπου το Spark αναλαμβάνει τον πλήρη έλεγχο τόσο των μεταδεδομένων όσο και των ίδιων των δεδομένων (Parquet Files). Όλες οι πληροφορίες σχετικά με τον πίνακα, συμπεριλαμβανομένης της δομής και της θέσης του, αποθηκεύονται στον εσωτερικό κατάλογο του Spark. Οι Managed Delta Tables είναι ο προεπιλεγμένος τύπος πίνακα στο Spark και χρησιμοποιείται όταν επιθυμούμε το Spark να χειρίζεται ολόκληρο τον κύκλο ζωής του πίνακα, από τη δημιουργία έως την αποθήκευση δεδομένων και την εκκαθάριση.

- External Delta Tables: Είναι οι πίνακες έχουν σχεδιαστεί για πρόσβαση σε δεδομένα που είναι αποθηκευμένα εκτός του ελέγχου του Spark. Σε έναν External Delta Table, το Spark διαχειρίζεται μόνο τα μεταδεδομένα στον εσωτερικό του κατάλογο, ενώ τα πραγματικά δεδομένα βρίσκονται σε ένα εξωτερικό σύστημα αποθήκευσης όπως το HDFS*, S3* ή μια σχεσιακή βάση δεδομένων. Οι External Delta Tables είναι ιδανικοί για σενάρια όπου τα δεδομένα μοιράζονται σε διαφορετικές εφαρμογές Spark ή όταν προκύπτει η ανάγκη να αξιοποιήσουμε υπάρχοντα

δεδομένα που είναι αποθηκευμένα σε εξωτερικές τοποθεσίες χωρίς να τα μετακινήσουμε ή να τα αντιγράψουμε [όπως παρουσιάζεται στο 13].

*HDFS: Το HDFS σημαίνει Hadoop Distributed File System. Το HDFS λειτουργεί ως ένα κατανεμημένο σύστημα αποθήκευσης αρχείων.

*S3: Η υπηρεσία Amazon Simple Storage Service (Amazon S3) είναι μια υπηρεσία αποθήκευσης αντικειμένων που προσφέρει κορυφαία επεκτασιμότητα, διαθεσιμότητα δεδομένων, ασφάλεια και απόδοση.

5.5 Πότε είναι απαραίτητη η χρησιμοποίηση των Delta Tables

1) Οι Delta Tables χρησιμοποιούνται για δεδομένα που ενημερώνονται συχνά, όπως δεδομένα ροής, δεδομένα συμβάντων και αρχεία καταγραφής εφαρμογών.

2) Μπορούν επίσης να χρησιμοποιηθούν για δεδομένα που δεν ενημερώνονται συχνά αλλά πρέπει να είναι προσβάσιμα σχεδόν σε πραγματικό χρόνο, όπως οικονομικά δεδομένα ή δεδομένα ενέργειας [όπως παρουσιάζεται στην βιβλιογραφία 11].

5.6 Διαφορές Delta Tables-Table SQL

Πίνακας Delta:

- ACID Transaction: Οι πίνακες Delta παρέχουν υποστήριξη συναλλαγών ACID (Atomicity, Consistency, Isolation, Durability). Αυτό σημαίνει ότι οι συναλλαγές στους πίνακες Delta είναι ατομικές και διατηρούν τη συνέπεια των δεδομένων, ακόμη και με την παρουσία ταυτόχρονων αναγνώσεων και εγγραφών.
- Schema Evolution: Οι Delta Tables υποστηρίζουν την εξέλιξη του σχήματος τους, επιτρέποντάς να τροποποιηθεί το σχήμα του πίνακα (προσθήκη, τροποποίηση ή διαγραφή στηλών) χωρίς να απαιτείται πλήρης επανεγγραφή του πίνακα.
- Time Travel: Παρέχουν δυνατότητες μεταβίβασης σε προηγούμενες καταστάσεις του πίνακα, επιτρέποντάς την αναζήτηση των δεδομένων του πίνακα σε συγκεκριμένη έκδοση ή ώρα. Αυτή η δυνατότητα μας επιτρέπει να εξερευνήσουμε και να επιστρέψουμε σε προηγούμενες εκδόσεις των δεδομένων.
- Αυξητικές αλλαγές δεδομένων: Οι πίνακες Delta χειρίζονται αποτελεσματικά τις αυξητικές αλλαγές δεδομένων. Χρησιμοποιούν ένα αρχείο καταγραφής συναλλαγών για να παρακολουθούν τις αλλαγές και οι τροποποιήσεις δεδομένων αποθηκεύονται ως αρχεία Delta Logs. Αυτό επιτρέπει πιο αποτελεσματικές ενημερώσεις και διαγραφές σε σύγκριση με τους παραδοσιακούς πίνακες.

Παραδοσιακός πίνακας SQL:

- ACID Transaction: Οι παραδοσιακοί πίνακες SQL υποστηρίζουν επίσης συναλλαγές, αλλά το επίπεδο υποστήριξης συναλλαγών μπορεί να διαφέρει ανάλογα με το σύστημα βάσης δεδομένων. Ορισμένες σχεσιακές βάσεις δεδομένων παρέχουν ισχυρές ιδιότητες ACID, ενώ άλλες μπορεί να έχουν διαφορετικά επίπεδα απομόνωσης.
- Schema Evolution: Η τροποποίηση του σχήματος ενός παραδοσιακού πίνακα SQL μπορεί να είναι μια πιο περίπλοκη λειτουργία. Μπορεί να περιλαμβάνει τη δημιουργία νέου πίνακα, τη μετεγκατάσταση δεδομένων και πιθανώς διακοπές λειτουργίας κατά τη διάρκεια της διαδικασίας.
- Time Travel: Οι περισσότερες παραδοσιακές βάσεις δεδομένων SQL δεν διαθέτουν ενσωματωμένες λειτουργίες μεταβίβασης σε παλαιότερες εκδόσεις των πινάκων τους. Για να επιτύχουμε παρόμοια λειτουργικότητα, χρειάζεται να διατηρούμε στην μνήμη μας αρκετά Back Ups του κάθε πίνακα.
- Αυξητικές αλλαγές δεδομένων: Οι παραδοσιακοί πίνακες χειρίζονται αυξητικές αλλαγές δεδομένων μέσω τυπικών λειτουργιών SQL (INSERT, UPDATE, DELETE). Ωστόσο, αυτές οι λειτουργίες ενδέχεται να είναι λιγότερο βελτιστοποιημένες για τροποποιήσεις δεδομένων μεγάλης κλίμακας σε σύγκριση με τα αρχεία δέλτα που χρησιμοποιούνται στους πίνακες Delta.

Κεφάλαιο 6: Συναλλαγές στους Delta Tables

Στο πρώτο μέρος της εργασίας, είχε γίνει μετατροπή του αρχείου nary_dynamic.csv σε Parquet file. Σε αυτό το σημείο λοιπόν θα χρησιμοποιηθεί ξανά για την δημιουργία ενός Delta Table στο Databricks, μπορεί να γίνει ακόμα πιο αποδοτικό στα ερωτήματα που του υποβάλλονται.

6.1 Τι είναι το Databricks;

Το Databricks είναι μια Cloud-Based πλατφόρμα-εργαλείο το οποίο χρησιμοποιείται κυρίως από μηχανικούς δεδομένων για την ασφαλή συλλογή, επεξεργασία και ανάλυση των δεδομένων που διαχειρίζονται. Αποτελεί σημαντική λύση για τις εταιρείες που επιθυμούν να διασφαλίζουν την ασφαλή αποθήκευση των ευαίσθητων δεδομένων τους, καθώς τα δεδομένα αποθηκεύονται στο Cloud Storage του Databricks (dbfs) με την ασφάλεια τους να την διαχειρίζεται ο ίδιος ο πάροχος (Databricks). Συνεπώς το Databricks είναι μία σπουδαία και ασφαλή επιλογή για την αποθήκευση και την επεξεργασία των δεδομένων κάθε εταιρίας [όπως περιγράφεται στην βιβλιογραφία 14].

6.2 Δημιουργία Delta Table

1ο Βήμα: Για αρχή θα πρέπει να μεταφερθεί το nary_dynamic.parquet αρχείο στο Databricks.

2ο Βήμα: Δημιουργείται μία βάση δεδομένων, για τον Delta πίνακα (είτε ορίζοντας την τοποθεσία μέσα στο File System, είτε όχι, η διαφορά μεταξύ των δύο είναι ότι όταν του οριστεί συγκεκριμένη τοποθεσία θα αποθηκεύει για τους πίνακες που ανήκουν στην συγκεκριμένη βάση τα αρχεία τους μέσα στην συγκριμένη τοποθεσία αλλιώς το Databricks έχει by default δική του τοποθεσία που τα εναποθέτει, για κάθε βάση ξεχωριστά).

3ο Βήμα: Παραγωγή του Delta πίνακα μέσα από το αρχείο.

```
CREATE TABLE IF NOT EXISTS my_db.delta_table_nary_dynamic
USING DELTA
AS
SELECT *
FROM parquet.`dbfs:/FileStore/nari_dynamic_prq.parquet`
```

*my_db: Η βάση που έχει δημιουργηθεί.

*«dbfs:/FileStore/nari_dynamic_prq.parquet»: η τοποθεσία αποθήκευσης του Paquet αρχείου.

6.3 Πληροφορίες Delta Table

Που όμως βρίσκονται όλα εκείνα τα αρχεία που περιέχει ένας Delta πίνακας, στα οποία έχει γίνει και πιο πάνω αναφορά;

Θα πρέπει αρχικά εφόσον δεν του έχει οριστεί τοποθεσία αποθήκευσης των αρχείων να αναζητηθεί που ακριβώς θα αποθηκεύει τα αρχεία και τα metadata του.

Με την εντολή **Describe Detail «table_name»** , λαμβάνεται αυτή την πληροφορία.

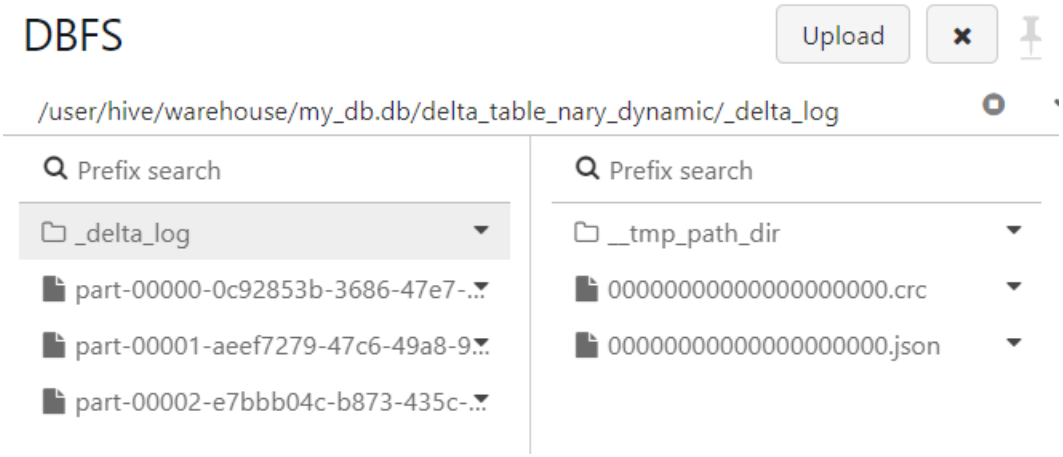
```
describe detail my_db.delta_table_nary_dynamic
```

_sqldf: pyspark.sql.dataframe.DataFrame = [format: string, id: string ... 11 more fields]

Table + New result table: ON Search

	format	id	name	description	location	createdAt	las
1	delta	8657632d-14e8-4824-a98b-6aa46d5b0b...	my_db.delta_table_nary_dyna...	null	dbfs:/user/hive/warehouse/my_db.db/delta_table_nary_dynamic	2024-04-27T14:46:35.565+00:...	2024-(

Άρα, έχοντας αυτή την πληροφορία μπορεί κανείς πλέον να περιηγηθεί μέσα στην τοποθεσία, για να δει τι ακριβώς περιέχει ένας τέτοιου είδους πίνακας από την αρχή της δημιουργίας του.



Στην παραπάνω εικόνα (τοποθεσία του Delta πίνακα) παρατηρείται το εξής:

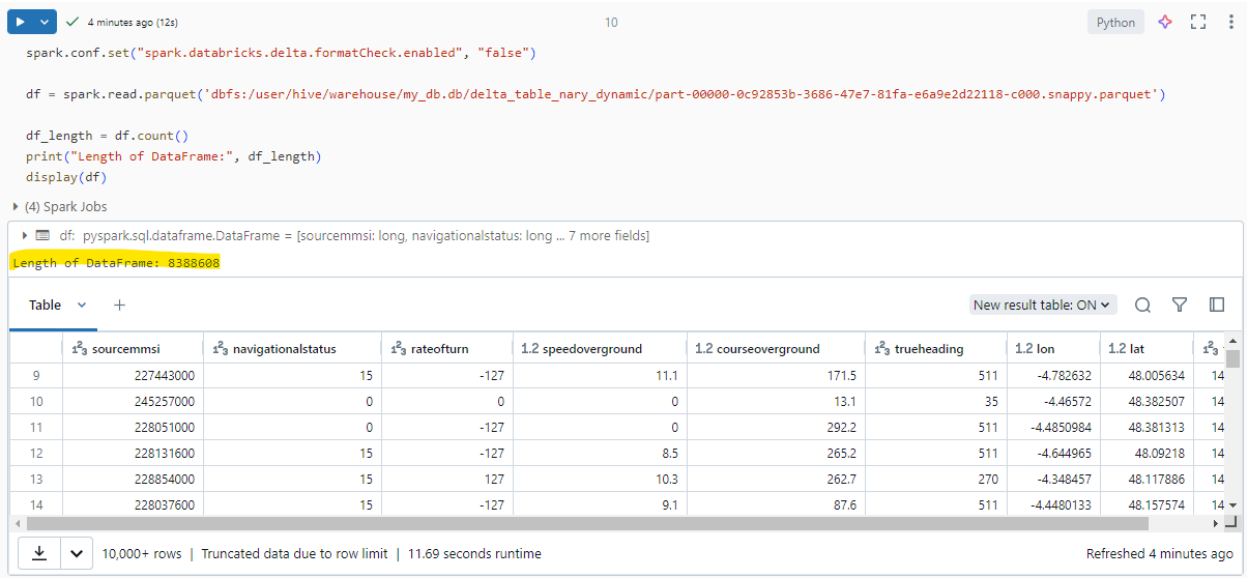
- 3 Parquet αρχεία
- 1 Φάκελο με όνομα _delta_log

Αρχεία Parquet

Τα αρχεία Parquet περιέχουν όλη την πληροφορία που είχε το αρχικό Parquet αρχείο, χωρισμένη σε επιμέρους αρχεία από την μηχανή.

Κάθε αρχείο Parquet περιέχει πληροφορία για μία η περισσότερες γραμμές τους πίνακα. Ο διαχωρισμός γίνεται από την μηχανή, ώστε για μεγάλα αρχεία πάνω του 1GB (όπως στην δική μας περίπτωση) να μπορέσει να μοιράσει την πληροφορία σε ίσα σε μέγεθος αρχεία, για να μπορεί πιο αποτελεσματικά να διαχειριστεί τον μεγάλο όγκο των δεδομένων που περιέχει, διαβάζοντας μικρότερα σε μέγεθος αρχεία, είτε για σκοπούς ερωτημάτων, είτε για σκοπούς συναλλαγών που υπόκεινται οι πίνακες.

1) part-00000-0c92853b-3686-47e7-81fa-e6a9e2d22118-c000.snappy.parquet



```
spark.conf.set("spark.databricks.delta.formatCheck.enabled", "false")

df = spark.read.parquet('dbfs:/user/hive/warehouse/my_db.db/delta_table_nary_dynamic/part-00000-0c92853b-3686-47e7-81fa-e6a9e2d22118-c000.snappy.parquet')

df_length = df.count()
print("Length of DataFrame:", df_length)
display(df)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [sourcemmsi: long, navigationalstatus: long ... 7 more fields]

Length of DataFrame: 8388608

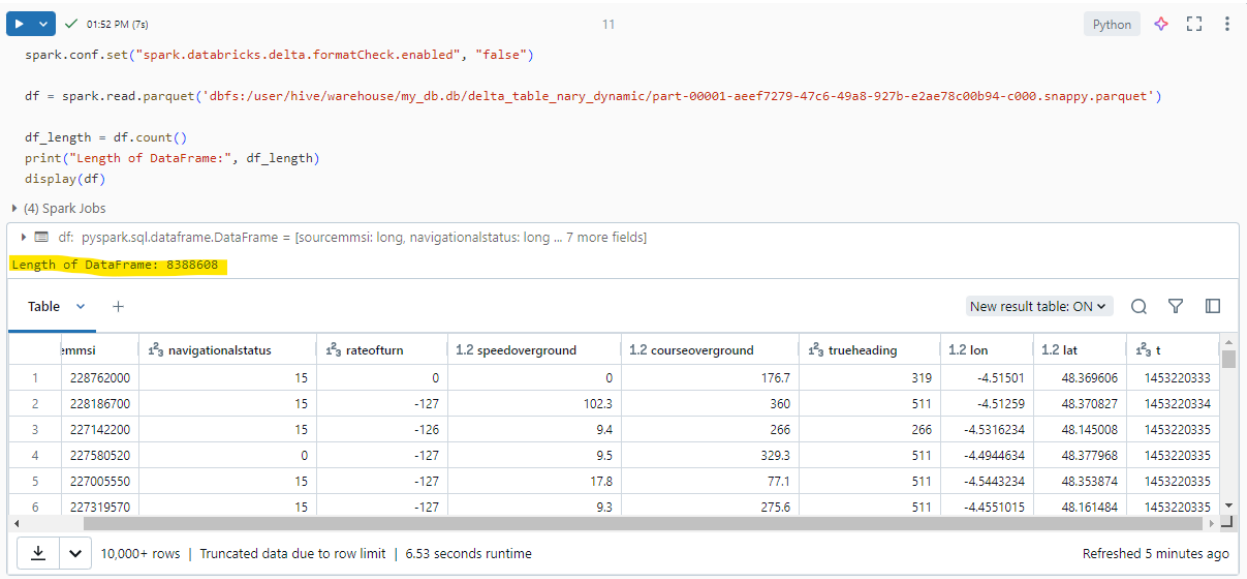
Table

	sourcemmsi	navigationalstatus	rateofturn	speedoverground	courseoverground	trueheading	lon	lat	
9	227443000	15	-127	11.1	171.5	511	-4.782632	48.005634	14
10	245257000	0	0	0	13.1	35	-4.46572	48.382507	14
11	228051000	0	-127	0	292.2	511	-4.4850984	48.381313	14
12	228131600	15	-127	8.5	265.2	511	-4.644965	48.09218	14
13	228854000	15	127	10.3	262.7	270	-4.348457	48.117886	14
14	228037600	15	-127	9.1	87.6	511	-4.4480133	48.157574	14

10,000+ rows | Truncated data due to row limit | 11.69 seconds runtime

Refreshed 4 minutes ago

2) part-00001-aeef7279-47c6-49a8-927b-e2ae78c00b94-c000.snappy.parquet



```
spark.conf.set("spark.databricks.delta.formatCheck.enabled", "false")

df = spark.read.parquet('dbfs:/user/hive/warehouse/my_db.db/delta_table_nary_dynamic/part-00001-aeef7279-47c6-49a8-927b-e2ae78c00b94-c000.snappy.parquet')

df_length = df.count()
print("Length of DataFrame:", df_length)
display(df)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [sourcemmsi: long, navigationalstatus: long ... 7 more fields]

Length of DataFrame: 8388608

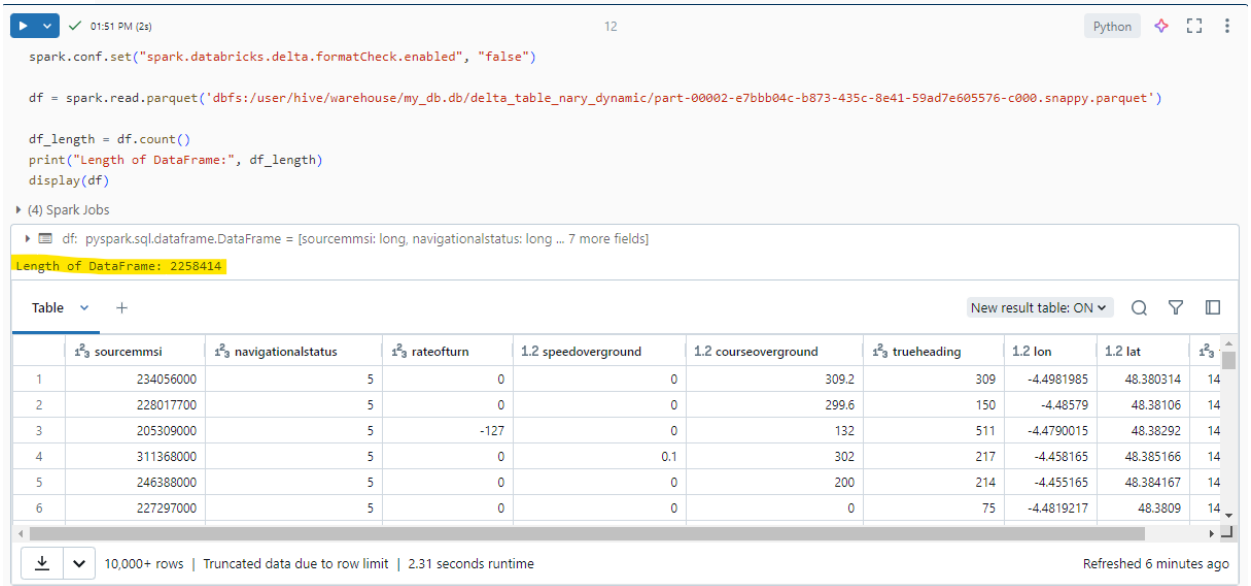
Table

	mmsi	navigationalstatus	rateofturn	speedoverground	courseoverground	trueheading	lon	lat	t
1	228762000	15	0	0	176.7	319	-4.51501	48.369606	1453220333
2	228186700	15	-127	102.3	360	511	-4.51259	48.370827	1453220334
3	227142200	15	-126	9.4	266	266	-4.5316234	48.145008	1453220335
4	227580520	0	-127	9.5	329.3	511	-4.4944634	48.377968	1453220335
5	227005550	15	-127	17.8	77.1	511	-4.5443234	48.353874	1453220335
6	227319570	15	-127	9.3	275.6	511	-4.4551015	48.161484	1453220335

10,000+ rows | Truncated data due to row limit | 6.53 seconds runtime

Refreshed 5 minutes ago

3) part-00002-e7bbb04c-b873-435c-8e41-59ad7e605576-c000.snappy.parquet



```
spark.conf.set("spark.databricks.delta.formatCheck.enabled", "false")

df = spark.read.parquet('dbfs:/user/hive/warehouse/my_db.db/delta_table_nary_dynamic/part-00002-e7bbb04c-b873-435c-8e41-59ad7e605576-c000.snappy.parquet')

df_length = df.count()
print("Length of DataFrame:", df_length)
display(df)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [sourcemmsi: long, navigationalstatus: long ... 7 more fields]

Length of DataFrame: 2258414

Table

	sourcemmsi	navigationalstatus	rateofturn	speedoverground	courseoverground	trueheading	lon	lat	
1	234056000	5	0	0	309.2	309	-4.4981985	48.380314	14
2	228017700	5	0	0	299.6	150	-4.48579	48.38106	14
3	205309000	5	-127	0	132	511	-4.4790015	48.38292	14
4	311368000	5	0	0.1	302	217	-4.458165	48.385166	14
5	246388000	5	0	0	200	214	-4.455165	48.384167	14
6	227297000	5	0	0	0	75	-4.4819217	48.3809	14

10,000+ rows | Truncated data due to row limit | 2.31 seconds runtime

Refreshed 6 minutes ago

Άρα λαμβάνοντας υπόψη τις υπογραμμισμένες επισημάνσεις στις οποίες διακρίνεται ο αριθμός των γραμμών που περιέχει κάθε Parquet αρχείο, παρατηρούμε ότι τα πρώτα δύο αρχεία έχουν τον ίδιο αριθμό γραμμών με αριθμό 8.388.608 και το τελευταίο αρχείο με 2.258.414 αριθμό γραμμών.

Την συγκεκριμένη τεχνική διαχωρισμού του ενός αρχείο σε μικρότερα, επιλέγει να χρησιμοποιήσει η μηχανή, ώστε να είναι σε θέση να απαντήσει όσο γίνεται πιο αποτελεσματικά στα ερωτήματα που θα της τεθούν, με τον διαχωρισμό των δεδομένων μας σε μικρότερα αρχεία, η πληροφορία διαμοιράζεται στα επιμέρους αρχεία, έχοντας κάθε αρχείο τα δικά του metadata, με την βοήθεια των οποίων η μηχανή τα χρησιμοποιεί ώστε να διαβάσει τα αρχεία που πραγματικά θα χρειαστεί, από το να είχε ένα αρχείο μόνο που να έπρεπε να διαβάσει.

Φάκελος _delta_log

Στον φάκελο μέσα υπάρχει ένας υποφάκελος ο οποίος χρησιμοποιείται από την μηχανή, όταν στον πίνακα συμβαίνουν Transactions.

Επίσης υπάρχουν αρχεία CRC και JSON, τα metadata δηλαδή που δημιουργεί ο πίνακας για κάθε συναλλαγή που υπόκεινται.

- JSON: Περιέχει πληροφορία για τα αρχεία, το όνομα του αρχείου/αρχεία Parquet που αναφέρεται, στο μέγεθος του, στον αριθμό των γραμμών που περιέχει κ.α.

Με μια πρώτη αρχική ματιά παρατηρούμε πως το αρχικό JSON αρχείο που έχει δημιουργηθεί περιέχει πληροφορίες τις πληροφορίες, όπως την εντολή με την οποία δημιουργήθηκε (**CREATE TABLE AS SELECT**), ορισμένες βασικές πληροφορίες για τον πίνακα που διακρίνονται με την κίτρινη επισήμανση, καθώς και πληροφορίες για τα αρχεία Parquet που περιέχει, όπως φαίνονται με την πράσινη υπογράμμιση.

- CRC: Είναι ένα αρχείο το οποίο χρησιμοποιείται για τον εντοπισμό σφαλμάτων στα δεδομένα κατά τη μετάδοση ή την αποθήκευση. Τα CRC αρχεία συμβάλλουν στη διασφάλιση της ακεραιότητας των αρχείων καταγραφής συναλλαγών.

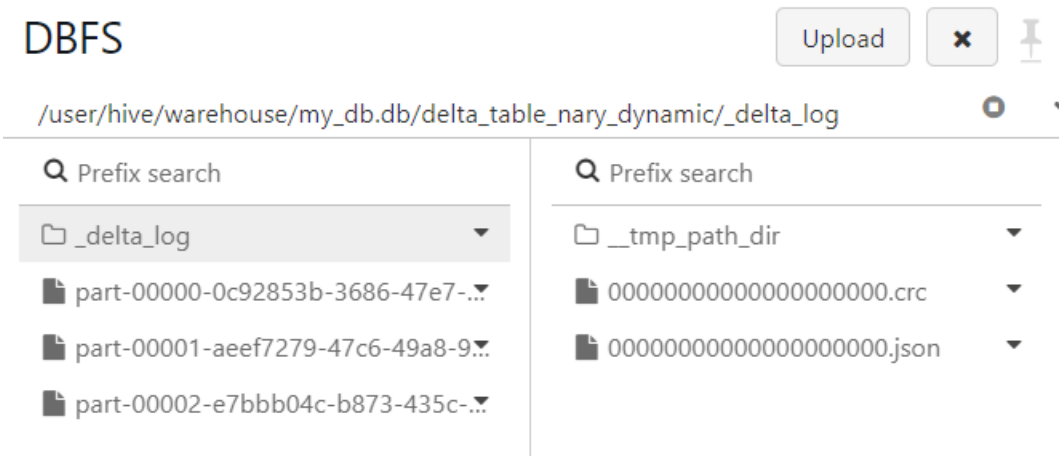
6.4 Συναλλαγές (Transactions) στους Delta Tables

Σε αυτό το σημείο θα αναλυθεί για κάθε πιθανή συναλλαγή που μπορεί να συμβεί για κάθε Delta πίνακα, τι ακριβώς συμβαίνει με τα Parquet αρχεία που αποθηκεύει.

Οι συναλλαγές που θα δούμε είναι οι εξής:

- INSERT
- UPDATE
- DELETE

Η τρέχουσα κατάσταση του πίνακα είναι η ακόλουθη:



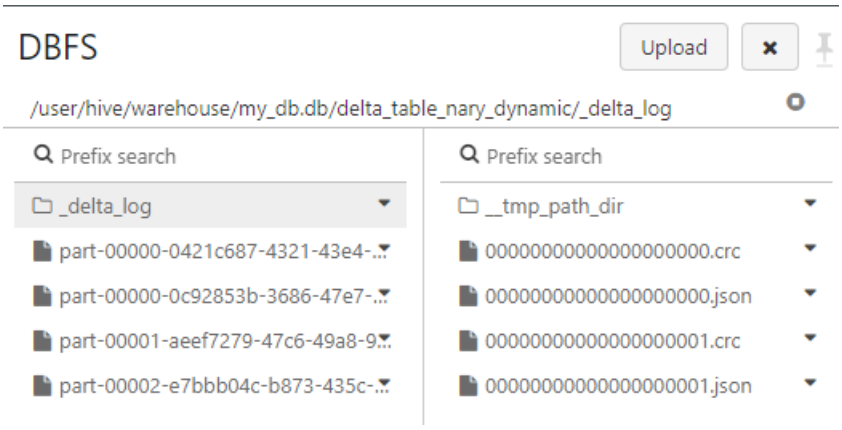
6.4.1 Insert

Όταν εκτελείται μια νέα λειτουργία εισαγωγής δεδομένων σε έναν πίνακα Delta (όπως φαίνεται στην Εικόνα 1 η συναλλαγή που εκτελείται στον πίνακα), η διαδικασία ξεκινά με την προετοιμασία δεδομένων, όπου τα δεδομένα τοποθετούνται σταδιακά, συνήθως σε μια προσωρινή θέση στην μνήμη του Databricks. Κατά τη διάρκεια αυτού του σταδίου, εκτελούνται έλεγχοι επικύρωσης δεδομένων και σχήματος για να διασφαλιστεί η συμβατότητα με το σχήμα του πίνακα Delta.

Εικόνα 1 - Εντολή Insert

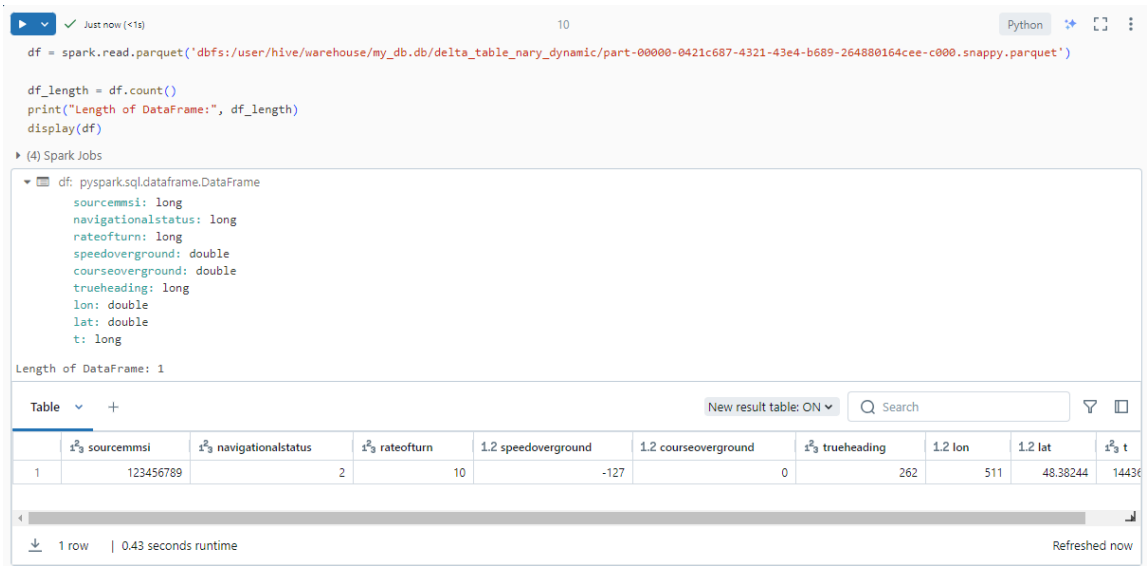
Στη συνέχεια, δημιουργείται ένα νέο αρχείο καταγραφής συναλλαγών JSON στον κατάλογο delta_log (Εικόνα 2 όπου εμφανίζεται η εσωτερική δομή του JSON αρχείου). Αυτό το αρχείο περιέχει μεταδεδομένα σχετικά με τη λειτουργία εισαγωγής της/των εγγραφής/εγγραφών. Σε κάθε αρχείο καταγραφής συναλλαγών εκχωρείται ένας μοναδικός αριθμός έκδοσης, διασφαλίζοντας μια διαδοχική εγγραφή όλων των συναλλαγών, όπως βλέπουμε παρακάτω (00000000000000001 στην δική μας περίπτωση).

Εικόνα 2 – Νέο αρχείο JSON μετά το Insert



Εικόνα 3 - Αρχεία Delta πίνακα μετά το Insert

Στη συνέχεια, τα δεδομένα εγγράφονται σε νέα αρχεία Parquet (Εικόνα 3 περιέχοντας τα αρχεία-μεταδεδομένα του πίνακα), τα οποία είναι βελτιστοποιημένα για αποτελεσματικές λειτουργίες ανάγνωσης και εγγραφής. Αυτά τα αρχεία ονομάζονται μοναδικά, για την αποφυγή συμφορήσεων. Μόλις δημιουργηθούν τα αρχεία δεδομένων, το Delta Lake εκτελεί μια λειτουργία ατομικής δέσμευσης. Αυτό το βήμα προσθέτει ατομικά τα νέα αρχεία δεδομένων στον χώρο αποθήκευσης του πίνακα Delta (Εικόνα 4 διαβάζοντας το νέο αρχείο παρατηρείται πως υπάρχει μέσα η νέα εγγραφή), διασφαλίζοντας ότι είτε προστίθενται όλα τα νέα αρχεία είτε δεν προστίθεται κανένα σε περίπτωση αποτυχίας. Το Delta Lake χρησιμοποιεί έναν αποτελεσματικό μηχανισμό ελέγχου συγχρονισμού για να χειρίζεται αποτελεσματικά τις ταυτόχρονες λειτουργίες εγγραφής.



Εικόνα 4 - Δεδομένα νέου Parquet αρχείου

Τα μεταδεδομένα του πίνακα Delta ενημερώνονται για να περιλαμβάνουν τα νέα αρχεία δεδομένων. Αυτή η ενημέρωση μεταδεδομένων περιλαμβάνει λεπτομέρειες όπως η λίστα των ενεργών αρχείων, οι εκδόσεις σχήματος και άλλες σημαντικές πληροφορίες.

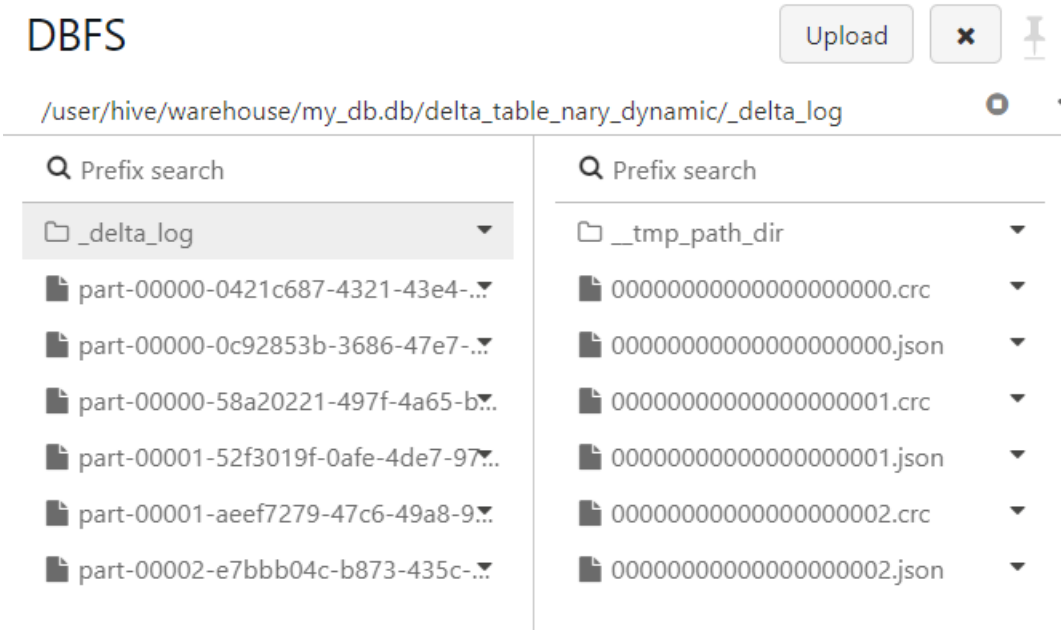
6.4.2 Update

Όταν εκτελείται μια λειτουργία ενημέρωσης (στην Εικόνα 5 φαίνεται η εντολή που έχει εφαρμοστεί) σε έναν πίνακα Delta, η διαδικασία ξεκινά με την αναγνώριση των δεδομένων μέσα από τα Parquet αρχεία. Γίνεται σάρωση του πίνακα για να εντοπίσει εγγραφές που ταιριάζουν με τα κριτήρια της εντολής.

Εικόνα 5 - Εντολή Update

Στη συνέχεια, δημιουργείται ένα νέο αρχείο καταγραφής συναλλαγών JSON στον κατάλογο delta_log (όπως φαίνεται στην Εικόνα 7 τα νέα πλέον αρχεία του πίνακα), το οποίο περιγράφει λεπτομερώς τη λειτουργία ενημέρωσης, συμπεριλαμβανομένων των επηρεαζόμενων αρχείων, των συνθηκών για την ενημέρωση και των νέων τιμών για τις ενημερωμένες εγγραφές (Εικόνα 6 δείχνοντας την εσωτερική δομή του αρχείου με την πληροφορία που περιέχει έπειτα το Update). Στη συνέχεια, η μηχανή Delta διαβάζει τα επηρεαζόμενα αρχεία και τροποποιεί τις εγγραφές στη μνήμη. Αυτές οι τροποποιημένες εγγραφές εγγράφονται σε νέα αρχεία Parquet, διασφαλίζοντας ότι τα αρχικά αρχεία παραμένουν αμετάβλητα έως ότου η δέσμευση είναι επιτυχής.

Εικόνα 6 – Νέο αρχείο JSON μετά το Update



Εικόνα 7 - Αρχεία Delta πίνακα μετά το Update

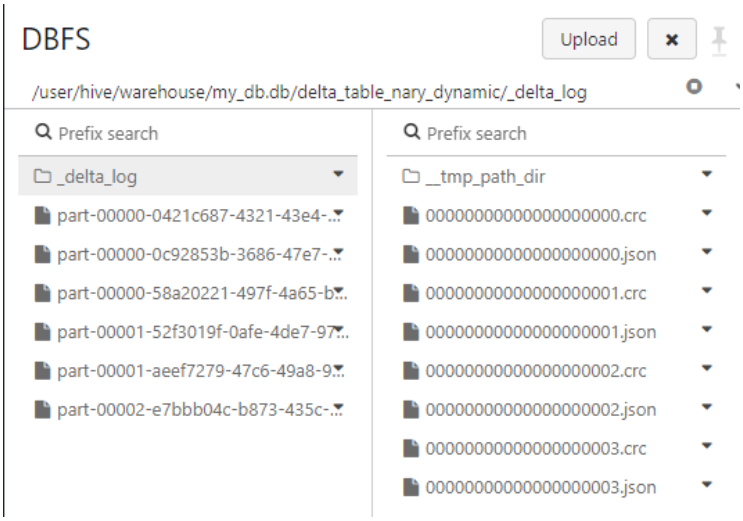
Οι παλιές εκδόσεις των αρχείων επισημαίνονται ως ανενεργά αρχεία (χωρίς απαραίτητα να έχουν διαγραφεί από την τοποθεσία του πίνακα) αλλά διατηρούνται μέχρι να εκτελεστεί μια λειτουργία εκκαθάρισης (VACUUM) όπου μόνο σε εκείνη την περίπτωση αφαιρούνται. Τέλος, τα μεταδεδομένα του πίνακα Delta ενημερώνονται για να αντικατοπτρίζουν τα νέα αρχεία, ενώ εξαιρούνται τα παλιά αρχεία. Η λειτουργία δέσμευσης οριστικοποιεί την ενημέρωση, καθιστώντας τις αλλαγές ορατές στις επόμενες λειτουργίες ανάγνωσης.

6.4.3 Delete

Όταν εκτελείται μια λειτουργία διαγραφής σε έναν πίνακα Delta, η διαδικασία ξεκινά με την αναγνώριση των δεδομένων προορισμού. Ο Delta πίνακας σαρώνει τον πίνακα για να βρει τις εγγραφές εκείνες που ταιριάζουν με τα κριτήρια διαγραφής.

Εικόνα 8 - Εντολή Delete

Στη συνέχεια δημιουργείται ένα νέο αρχείο καταγραφής συναλλαγών JSON στον κατάλογο delta_log (η Εικόνα 9 αναδεικνύει την νέα εικόνα των αποθηκευμένων αρχείων του πίνακα έχοντας προστεθεί μόνο μεταδεδομένα για αυτή την συναλλαγή), το οποίο τεκμηριώνει τη λειτουργία διαγραφής, συμπεριλαμβανομένων των επηρεαζόμενων αρχείων και των συνθηκών διαγραφής (όπως φαίνεται Εικόνα 10 διαβάζοντας το JSON αρχείο). Ο πίνακας διαβάζει τα επηρεαζόμενα αρχεία και επισημαίνει τις εγγραφές που πρέπει να διαγραφούν. Αυτές οι επισημασμένες εγγραφές διαγράφονται αλλά διατηρούνται φυσικά ως ανενεργά αρχεία μέχρι να εκτελεστεί μια λειτουργία καθαρισμού «VACUUM», όπου μετέπειτα αφαιρούνται κανονικά από την τοποθεσία αποθήκευσης του πίνακα.



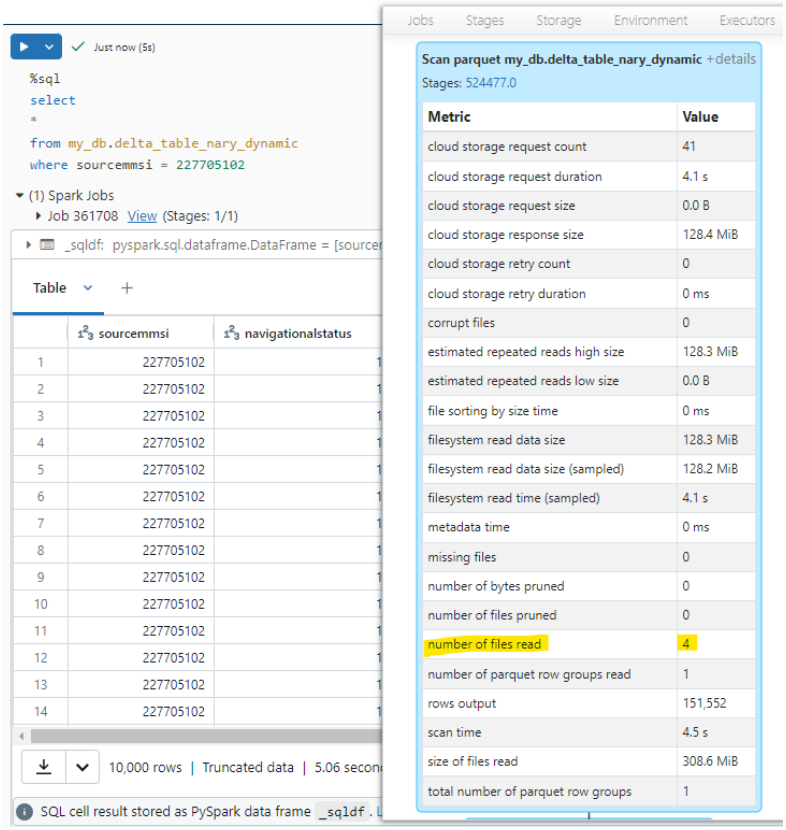
Εικόνα 9 - Αρχεία Delta πίνακα μετά το Delete

Εικόνα 10 - Νέο αρχείο JSON μετά το Delete

6.5 Inactive Parquet αρχεία

Η λειτουργικότητα των Delta Tables να καθιστούν σε ανενεργή (inactive) κατάσταση τα αρχεία που περιέχονται μέσα σε αυτά, μπορεί να προκαλέσει προβληματισμό για αυτό λοιπόν σε αυτό το σημείο μπορεί να εξακριβωθεί δημιουργώντας ένα απλό ερώτημα για τον πίνακα ώστε να διευκρινιστεί ο αριθμός των Parquet αρχείων που διαβάζει.

Πιο συγκεκριμένα,



Όπως φαίνεται παραπάνω για να απαντηθεί το ερώτημα, έχουν διαβαστεί τέσσερα αρχεία Parquet, αυτό άλλωστε ήταν το προσδοκώμενο, καθώς από το Update και τα Delete έχουν αδρανοποιηθεί τα δύο από τα έξη αρχεία που υπάρχουν στην τοποθεσία του πίνακα.

<u>Πως όμως αυτά τα αρχεία διαγράφονται;</u>

Η φυσική διαγραφή των ανενεργών αρχείων συμβαίνει κατά τη λειτουργία VACUUM, η οποία μπορεί να προγραμματιστεί να εκτελείται περιοδικά. Αυτή η λειτουργία ελευθερώνει χώρο αποθήκευσης αφαιρώντας όλα αρχεία που δεν χρειάζονται πλέον.

Κεφάλαιο 7: Βελτιστοποίηση Delta Tables

Σε αυτό το κεφάλαιο θα αναλυθούν οι τεχνικές βελτιστοποίησης των Delta Tables.

Μερικές από τις πιο σημαντικές τεχνικές βελτιστοποίησης είναι οι εξής:

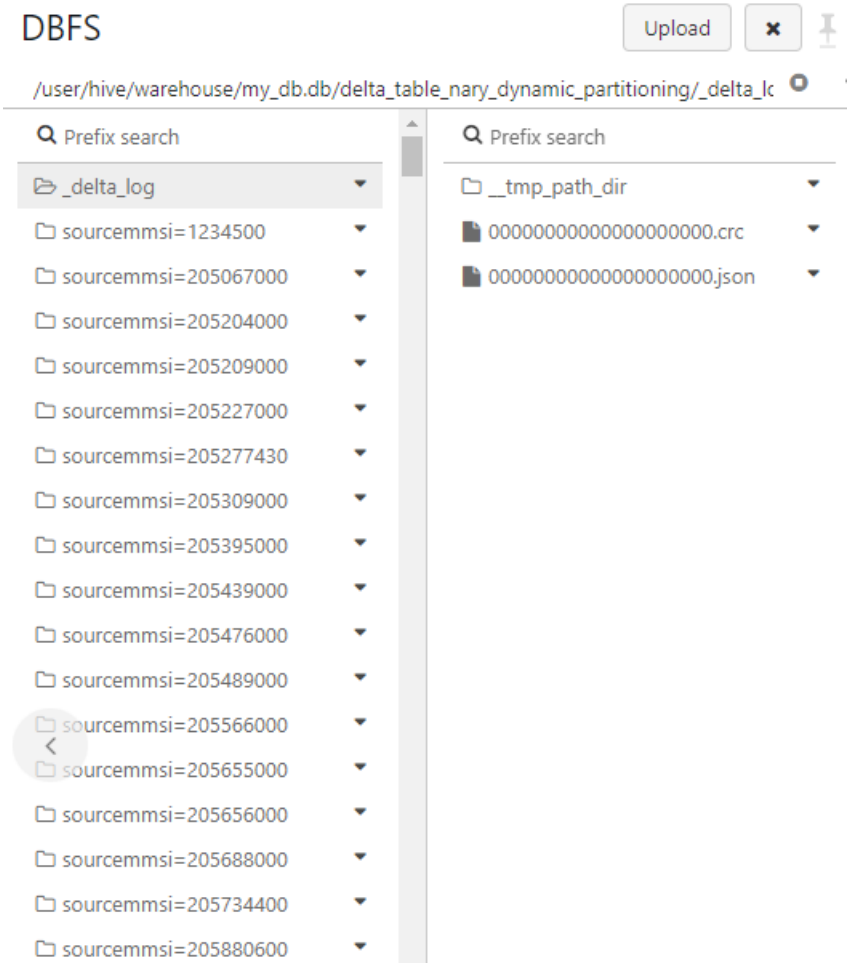
- Partitioning
- Optimize
- Z-Ordering

7.1 Partitioning

Το Partitioning είναι μια τεχνική που χρησιμοποιείται για τη διαίρεση ενός μεγάλου όγκου δεδομένων σε επιμέρους, κάνοντας τα πιο διαχειρίσιμα, με τον διαχωρισμό να γίνεται με βάση των ίδιων τιμών σε μία ή περισσότερες στήλες. Κάθε ξεχωριστή τοποθεσία μέσα στον πίνακα αντιπροσωπεύει ένα υποσύνολο δεδομένων και αποθηκεύεται σε ξεχωριστά paths (όπως φαίνεται στην Εικόνα 11 του πίνακα που έχει υποστεί την συγκεκριμένη τεχνική), διευκολύνοντας τη βελτιωμένη οργάνωση των δεδομένων και την απόδοση των ερωτημάτων.

Αυτή η μέθοδος είναι ιδιαίτερα αποτελεσματική για δεδομένα χρονοσειράς ή άλλα σενάρια όπου μια συγκεκριμένη στήλη χρησιμοποιείται συχνά για φιλτράρισμα. Τα νέα δεδομένα κατευθύνονται στον κατάλληλο κατάλογο διαμερισμάτων με βάση την τιμή του στη στήλη του διαμερίσματος. Αυτή η οργάνωση επιτρέπει στα ερωτήματα που φιλτράρουν στη στήλη που έχει γίνει ο διαχωρισμός να έχουν πρόσβαση μόνο στις σχετικές τοποθεσίες, μειώνοντας σημαντικά τον όγκο των δεδομένων που σαρώνονται και διαβάζονται βελτιώνοντας την απόδοση των ερωτημάτων. Μία απεικόνιση ενός Delta

πίνακα που του έχει εφαρμοστεί η συγκεκριμένη τεχνική είναι η ακόλουθη [όπως αναγράφεται στην 15η βιβλιογραφία].



Εικόνα 11 - Τοποθεσία Delta πίνακα μετά το Partitioning

Ένα σύνολο υποφακέλων λοιπόν για κάθε ξεχωριστή (Distinct) τιμή της συγκεκριμένης στήλης (στην συγκεκριμένη περίπτωση την στήλη sourcemmsi) που επιλέξαμε να εφαρμόσουμε την τεχνική του Partitioning.

7.2 Optimize

Η Optimize τεχνική εστιάζει στη βελτίωση της απόδοσης των ερωτημάτων και των συναλλαγών μέσω διαφόρων ενεργειών. Όπως και παραπάνω πολλές φορές τα δεδομένα εισαγωγής που εγγράφονται σε έναν πίνακα Delta, συχνά καταλήγουν σε πολλά μικρά αρχεία. Αυτά τα μικρά αρχεία μπορεί να οδηγήσουν σε αναποτελεσματικότητα της απόδοσης, καθώς η ανάγνωση πολλών μικρών αρχείων είναι πιο αργή σε σύγκριση με την ανάγνωση λιγότερων μεγάλων αρχείων. Η εντολή OPTIMIZE το αντιμετωπίζει, συμπυκνώνοντας αυτά τα μικρά αρχεία σε λιγότερα, μεγαλύτερα. Αυτή η διαδικασία συμπίεσης μειώνει τα γενικά έξοδα που σχετίζονται με την ανάγνωση πολλών μικρών αρχείων, βελτιώνοντας έτσι τη συνολική απόδοση ανάγνωσης.

Εκτός από τη συμπίεση αρχείων, οι Delta πίνακες χρησιμοποιούν την παράκαμψη δεδομένων, μια τεχνική που αξιοποιεί στατιστικά που είναι αποθηκευμένα στα μεταδεδομένα των αρχείων, όπως ελάχιστες και μέγιστες τιμές, για να παρακάμψει αρχεία που δεν ταιριάζουν με τα ερωτήματα. Αυτό βελτιώνει περαιτέρω την απόδοση των ερωτημάτων αποφεύγοντας τις περιττές σαρώσεις αρχείων.

Τα οφέλη της βελτιστοποίησης περιλαμβάνουν μειωμένες λειτουργίες I/O, καθώς μεγαλύτερα αρχεία σημαίνουν λιγότερες λειτουργίες ανάγνωσης και ταχύτερη εκτέλεση ερωτημάτων λόγω παράβλεψης δεδομένων [σύμφωνα με την αναφορά της 15^{ης} βιβλιογραφίας].

7.3 Z-Ordering

Το Z-Ordering είναι μια εξελιγμένη τεχνική ομαδοποίησης που μπορεί να εφαρμοστεί στους συγκεκριμένους πίνακες και έχει σχεδιαστεί για να εντοπίζει σχετικά δεδομένα μέσα στο ίδιο σύνολο αρχείων, βελτιστοποιώντας την απόδοση των ερωτημάτων, ιδιαίτερα για ερωτήματα που περιλαμβάνουν πολλαπλές στήλες. Με την εφαρμογή της ως αποτέλεσμα να έχουμε σειρές με

παρόμοιες τιμές στις καθορισμένες στήλες που εφαρμόζεται και αποθηκεύονται η μία κοντά στην άλλη μέσα στα αρχεία Parquet. Κατά αυτό τον τρόπο ενισχύεται η αποτελεσματικότητα των ερωτημάτων, καθώς χρειάζεται να διαβαστούν λιγότερα αρχεία για την ανάκτηση των σχετικών δεδομένων. Κατά αυτό τον τρόπο οδηγούμαστε σε ταχύτερη απόδοση. Τα πλεονεκτήματα του Z-Ordering περιλαμβάνουν βελτιωμένη αναζήτηση δεδομένων κάνοντας μας τα ερωτήματα πιο αποδοτικά και γρήγορα [όπως αναδεικνύεται στην βιβλιογραφία 15].

Σε αυτό το σημείο υπάρχει ένα παράδειγμα για την καλύτερη κατανόηση του τι ακριβώς συμβαίνει πίσω στα αρχεία με τις παραπάνω τεχνικές βελτιστοποίησης.

Έστω ο παρακάτω πίνακας:

Employee Delta Table

File 1

Emp_id	Emp_name	Salary	Active
111	Michael	5000	Y
555	Kevin	7000	Y

File 2

Emp_id	Emp_name	Salary	Active
333	David	4000	Y
999	Peter	3000	Y

File 3

Emp_id	Emp_name	Salary	Active
222	Nancy	5000	Y
444	Tomas	6000	Y

File 4

Emp_id	Emp_name	Salary	Active
333	David	2000	N
888	Shane	3000	N

File 5

Emp_id	Emp_name	Salary	Active
555	Kevin	3000	N
888	Shane	4500	Y

File 6

Emp_id	Emp_name	Salary	Active
444	Tomas	2500	N
999	Peter	2000	N

Ο οποίος περιέχει έξη Parquet αρχεία.

Με το να εφαρμοστεί η απλή εντολή Optimize θα προκύψει το παρακάτω αποτέλεσμα:

Optimized Employee Table

File 1

Emp_id	Emp_name	Salary	Active
111	Michael	5000	Y
555	Kevin	7000	Y
333	David	4000	Y
999	Peter	3000	Y

File 2

Emp_id	Emp_name	Salary	Active
222	Nancy	5000	Y
444	Tomas	6000	Y
333	David	2000	N
888	Shane	3000	N

File 3

Emp_id	Emp_name	Salary	Active
555	Kevin	3000	N
888	Shane	4500	Y
444	Tomas	2500	N
999	Peter	2000	N

select * from employee where emp_id =444	1,2,3
select * from employee where emp_id <333	1,2

File	Min	Max
File1	111	999
File2	222	888
File3	444	999

Με την συμπύκνωση των αρχείων θα δημιουργηθούν τρία νέα αρχεία.

Εφαρμόζοντας τα ερωτήματα στην παραπάνω φωτογραφία είναι εμφανές πως για emp_id=444, η μηχανή θα χρειαστεί να διαβάσει και τα τρία αρχεία. Για το πρώτο αρχείο όμως παρόλο που θα το διαβάσει δεν θα πάρει την πληροφορία που θέλει, άρα η ενέργεια αυτή καθίσταται ανούσια και αρκετά χρονοβόρα όταν ο πίνακας μας θα έχει πάρα πολλά αρχεία.

Εφαρμόζοντας την τεχνική του Z-Ordering βλέπουμε πως αυτό αποφεύγεται.

Z-Ordered Employee Table

File 1

Emp_id	Emp_name	Salary	Active
111	Michael	5000	Y
222	Nancy	5000	Y
333	David	4000	Y
333	David	2000	N

File 2

Emp_id	Emp_name	Salary	Active
444	Tomas	6000	Y
444	Tomas	2500	N
555	Kevin	7000	Y
555	Kevin	3000	N

File 3

Emp_id	Emp_name	Salary	Active
888	Shane	3000	N
888	Shane	4500	Y
999	Peter	3000	Y
999	Peter	2000	N

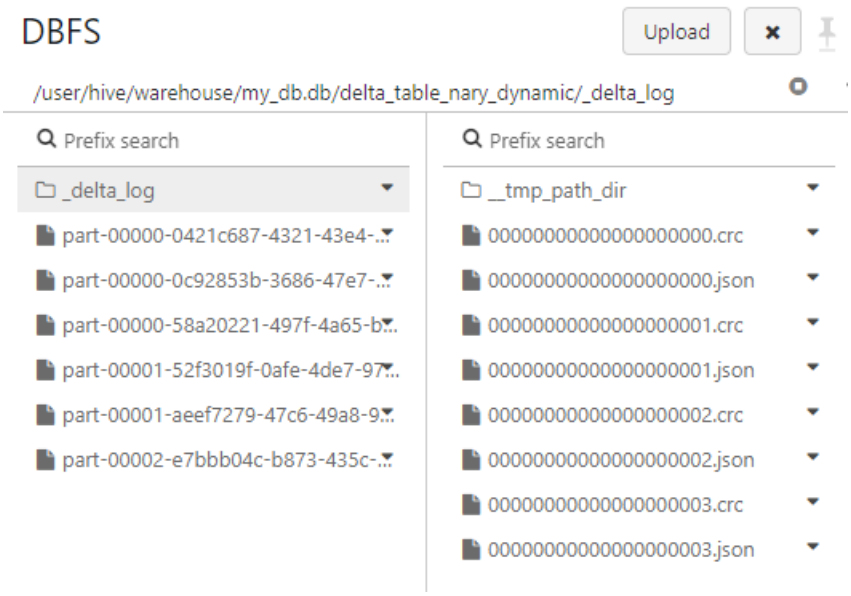
select * from employee where emp_id =444	2
select * from employee where emp_id <333	1

File	Min	Max
File1	111	333
File2	444	555
File3	888	999

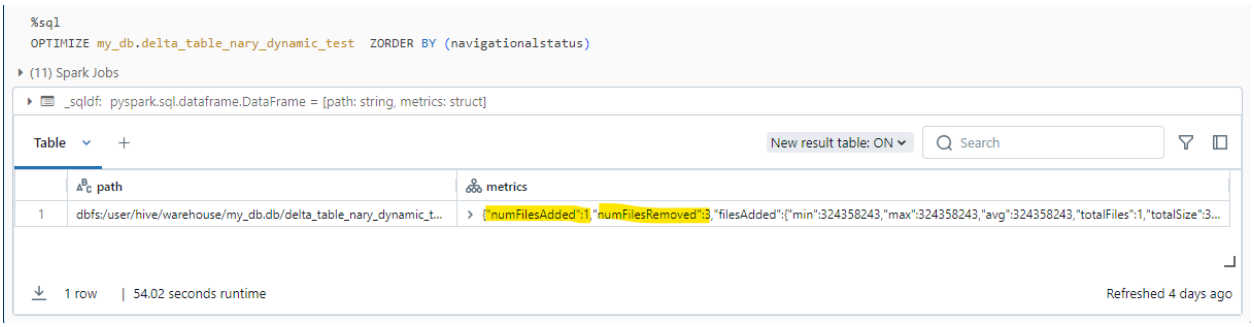
Πλέον η μηχανή για emp_id=444 θα διαβάσει μόνο το δεύτερο αρχείο το οποίο έχει και την πληροφορία που πρέπει να επιστρέψει. Άρα η αναζήτηση πλέον γίνεται πιο αποτελεσματική και γρήγορη.

Στο σημείο αυτό θα εφαρμοστεί η τεχνική του Z-Ordering στον πίνακα που έχει δημιουργηθεί.

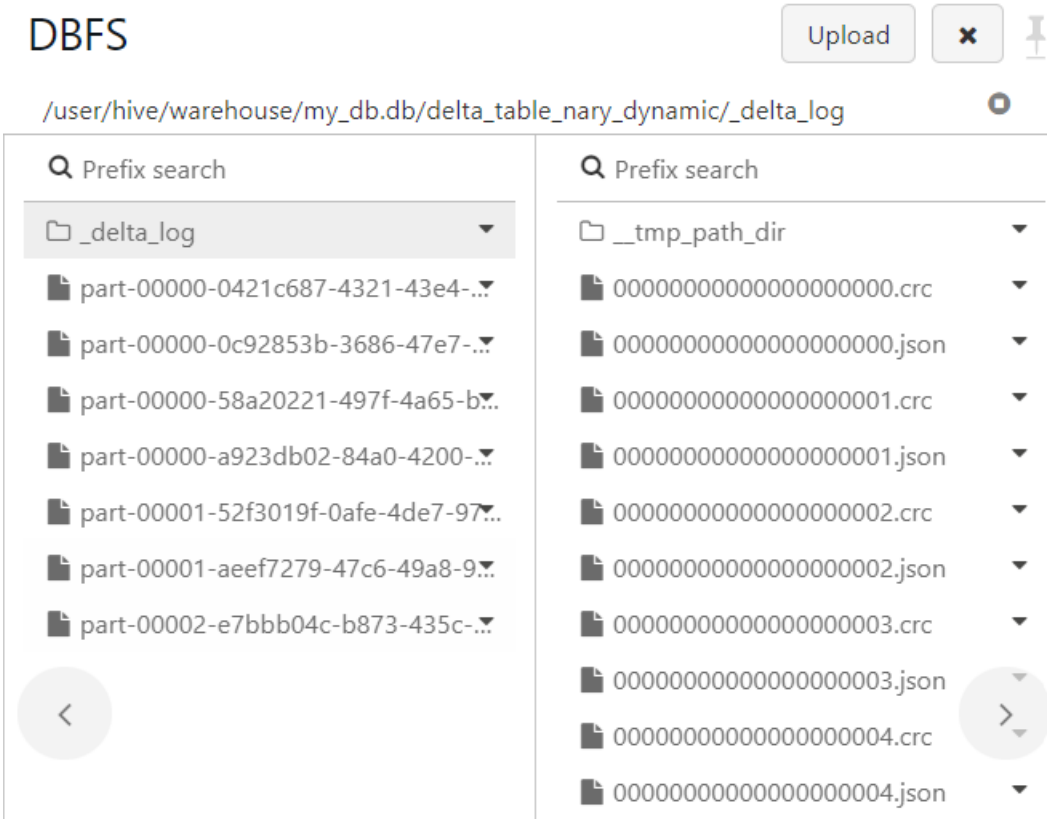
Η παρούσα κατάσταση του πίνακα είναι η εξής:



Υπάρχουν έξη στον αριθμό Parquet files (εκ των οποίων τέσσερα αρχεία είναι ενεργά και δύο είναι αδρανοποιημένα, και τρία στον αριθμό ζευγάρια CRC,JSON.



Όπως φαίνεται από την παραπάνω φωτογραφία έχουν αφαιρεθεί (όπως έχει επισυμανθεί και πιο πάνω με τον όρο «αφαιρεθεί» εννοείται πως έχουν αδρανοποιηθεί τα συγκεκριμένα αρχεία) τρία αρχεία από τα τέσσερα που ήταν ενεργά και έχει προστεθεί ένα συμπυκνωμένο.



Πράγματι διακρίνουμε εφτά αρχεία Parquet και ένα επιπλέον ζευγάρι metadata.

7.4 Πως λειτουργεί η μηχανή στα Queries;

Τι ακριβώς συμβαίνει πίσω από την μηχανή όταν της ανατίθενται ένα ερώτημα σε Delta πίνακα.

Όταν της ανατίθενται ένα ερώτημα η μηχανή για να επιστρέψει το πλήθος των δεδομένων του πίνακα πάνω στο εκάστοτε ερώτημα, διαβάζει όλα τα ενεργά (active) Parquet αρχεία που υπάρχουν μέσα στην τοποθεσία του πίνακα.

Γίνεται αντιληπτό πως σε μια συνεχόμενη και καθημερινή διαδικασία ETL (Extract Transform Load), όπου ο κάθε πίνακας Delta δέχεται Parquet αρχεία με συναλλαγές (Insert,Update,Delete όπως δείξαμε και πιο πάνω), κατά το πέρας του χρόνου συσσωρεύονται πολλά τέτοιου είδους αρχεία που καλείται να διαβάσει η μηχανή.

Το συγκεκριμένο λοιπόν, αποτελεί πρόβλημα για την αποδοτικότητα των Delta πινάκων, ως προς την ταχύτητα επιστροφής των δεδομένων.

7.4.1 Στόχος

Έτσι λοιπόν ο στόχος ο οποίος γεννιέται μέσα από τα παραπάνω είναι το πως θα μπορεί να βοηθηθεί η μηχανή, ώστε να διαβάζει λιγότερα αρχεία Parquet και εν τέλη να μπορέσει να επιστρέψει και πιο γρήγορα το επιθυμητό αποτέλεσμα.

Αυτό φυσικά μπορεί να επιτευχθεί με την εντολή OPTIMIZE που έχει περιγράφει παραπάνω στην υπό ενότητα 7.2, ή και φυσικά με την Z-ORDER που απλά είναι μία επέκταση της OPTIMIZE.

7.4.2 Δημιουργία Delta πίνακα με πολλαπλά Transactions (INSERT,UPDATE)

Οι πίνακες έχουν υποστεί πολλές συναλλαγές INSERT, UPDATE μέσω ενός κώδικα που έχει υλοποιηθεί για τον συγκεκριμένο σκοπό.

INSERTS Transactions:

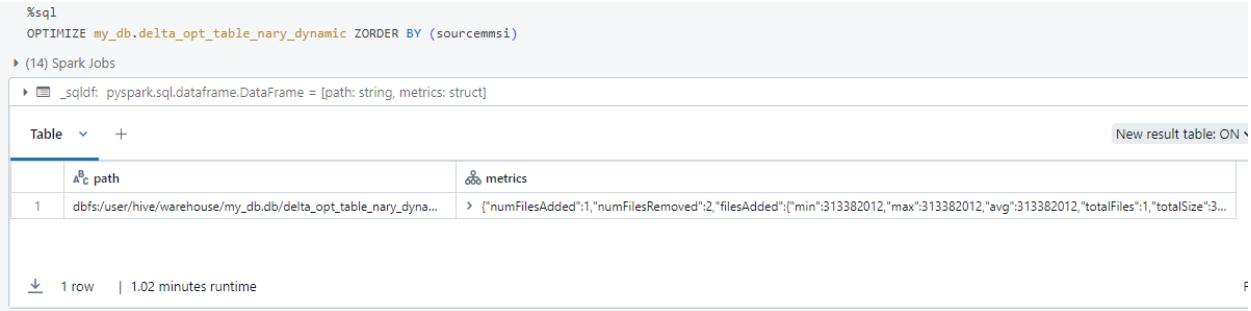
Ο κώδικας για τα INSERTS Transactions δημιουργεί τυχαίες τιμές για κάθε στήλη μεταξύ του εύρους MIN-MAX της εκάστοτε στήλης και την εφαρμόζει έπειτα και στους δύο πίνακες.

UPDATE Transactions:

Ο κώδικας για τα UPDATES Transactions επιλέγει μια τυχαία στήλη και μια τυχαία τιμή της συγκεκριμένης στήλης με βάση το INDEX της τιμής και την ορίζει με 0.

Στόχος είναι μέσα από απλά πολλαπλά Transactions, να προκύψουν πολλά αρχεία Parquet.

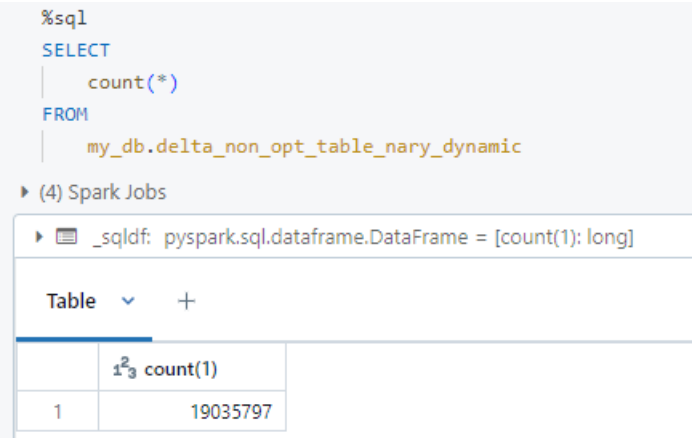
Κατά την ολοκλήρωση του κώδικα με την εφαρμογή των Transactions και στους δύο πίνακες εφαρμόζουμε την εντολή OPTIMIZED στον πίνακα delta_opt_table_nary_dynamic.

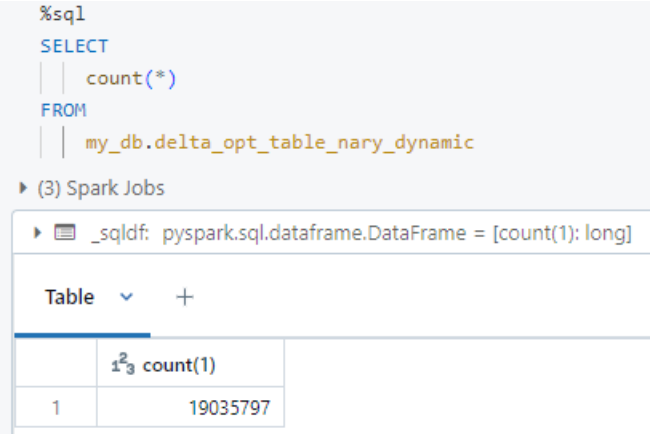


Οι πίνακες είναι οι εξής:

- delta_non_opt_table_nary_dynamic
- delta_opt_table_nary_dynamic

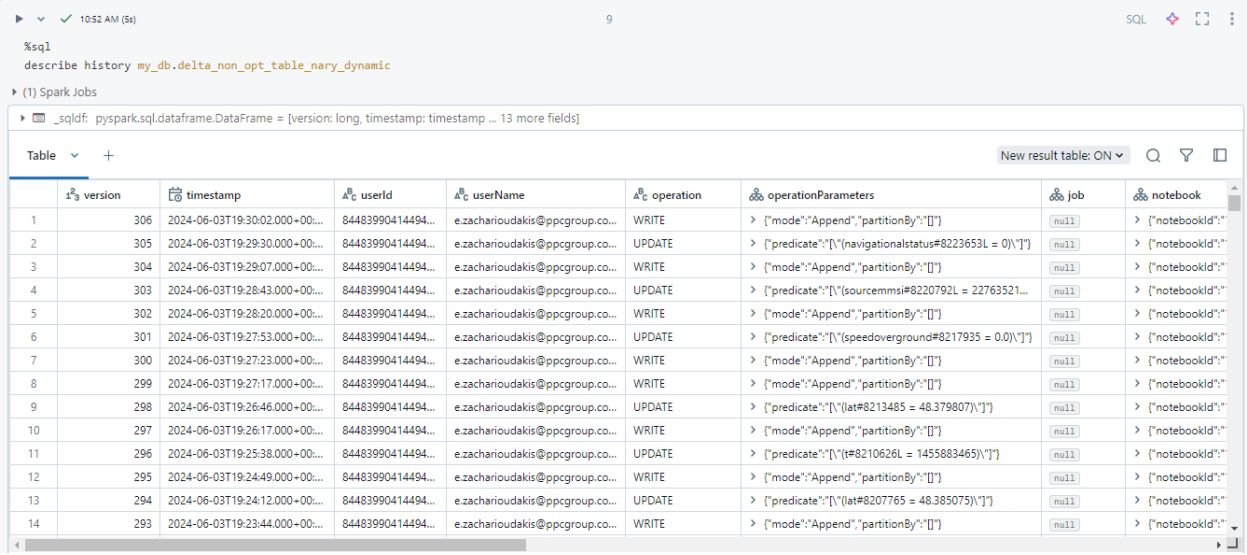
Για αρχή θα αποτυπωθεί το πλήθος του κάθε πίνακα.





Και οι δύο πίνακες έχουν τον ίδιο αριθμό εγγραφών.

Στην συνέχεια, τα versions του κάθε πίνακα (κάθε version είναι και μία αλλαγή που έχει υποστεί ο πίνακας, είτε INSERT WRITE όπως απεικονίζεται, είτε UPDATE όπως φαίνεται παρακάτω) είναι τα εξής:



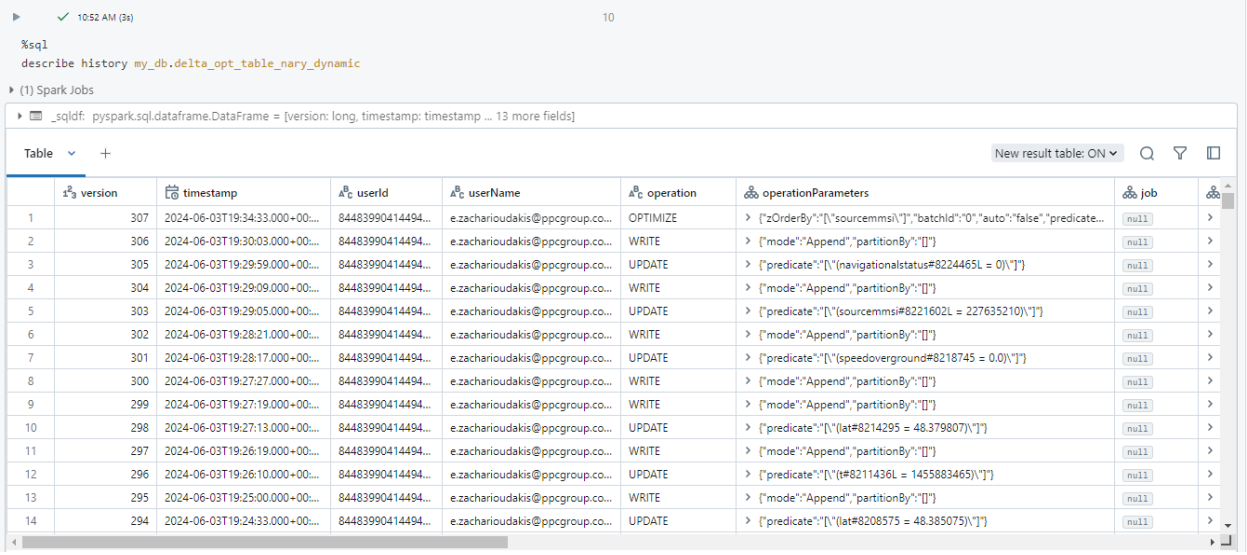
```
%sql
describe history my_db.delta_non_opt_table_nary_dynamic
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table

	version	timestamp	userId	userName	operation	operationParameters	job	notebook
1	306	2024-06-03T19:30:02.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
2	305	2024-06-03T19:29:30.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(navigationalstatus#8223653L = 0)\"]"}	null	{"notebookId":"
3	304	2024-06-03T19:29:07.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
4	303	2024-06-03T19:28:43.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(sourcemmsi#8220792L = 22763521...	null	{"notebookId":"
5	302	2024-06-03T19:28:20.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
6	301	2024-06-03T19:27:53.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(speedoverground#8217935 = 0.0)\"]"}	null	{"notebookId":"
7	300	2024-06-03T19:27:23.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
8	299	2024-06-03T19:27:17.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
9	298	2024-06-03T19:26:46.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(lat#8213485 = 48.379807)\"]"}	null	{"notebookId":"
10	297	2024-06-03T19:26:17.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
11	296	2024-06-03T19:25:38.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(t#8210626L = 1455883465)\"]"}	null	{"notebookId":"
12	295	2024-06-03T19:24:49.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"
13	294	2024-06-03T19:24:12.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(lat#8207765 = 48.385075)\"]"}	null	{"notebookId":"
14	293	2024-06-03T19:23:44.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	{"notebookId":"



```
%sql
describe history my_db.delta_opt_table_nary_dynamic
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table

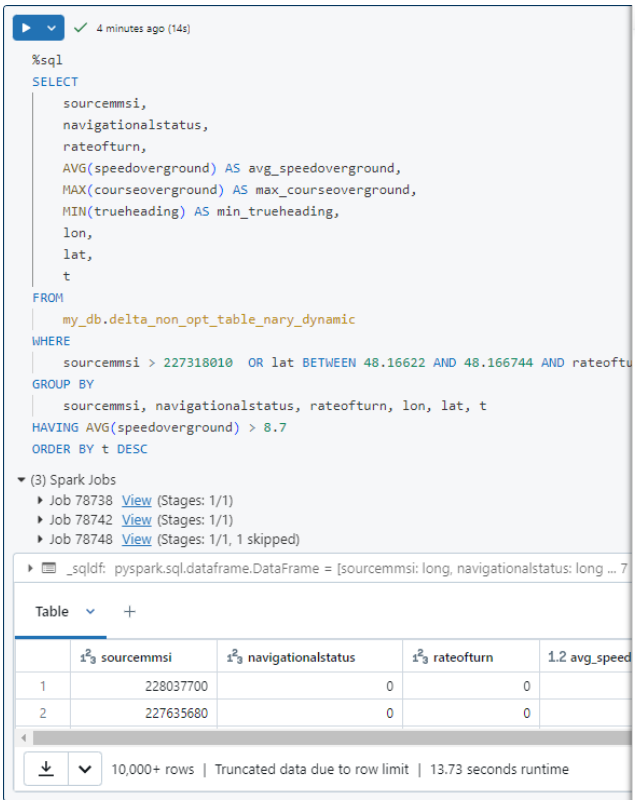
	version	timestamp	userId	userName	operation	operationParameters	job	
1	307	2024-06-03T19:34:33.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	OPTIMIZE	{"zOrderBy":"[\"sourcemmsi\"]","batchId":"0","auto":"false","predicate...	null	
2	306	2024-06-03T19:30:03.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
3	305	2024-06-03T19:29:59.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(navigationalstatus#8224465L = 0)\"]"}	null	
4	304	2024-06-03T19:29:09.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
5	303	2024-06-03T19:29:05.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(sourcemmsi#8221602L = 227635210)\"]"}	null	
6	302	2024-06-03T19:28:21.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
7	301	2024-06-03T19:28:17.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(speedoverground#8218745 = 0.0)\"]"}	null	
8	300	2024-06-03T19:27:27.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
9	299	2024-06-03T19:27:19.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
10	298	2024-06-03T19:27:13.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(lat#8214295 = 48.379807)\"]"}	null	
11	297	2024-06-03T19:26:19.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
12	296	2024-06-03T19:26:10.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(t#8211436L = 1455883465)\"]"}	null	
13	295	2024-06-03T19:25:00.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	WRITE	{"mode":"Append","partitionBy":"[]"}	null	
14	294	2024-06-03T19:24:33.000+00:...	84483990414494...	e.zacharioudakis@ppcgroup.co...	UPDATE	{"predicate":"[\"(lat#8208575 = 48.385075)\"]"}	null	

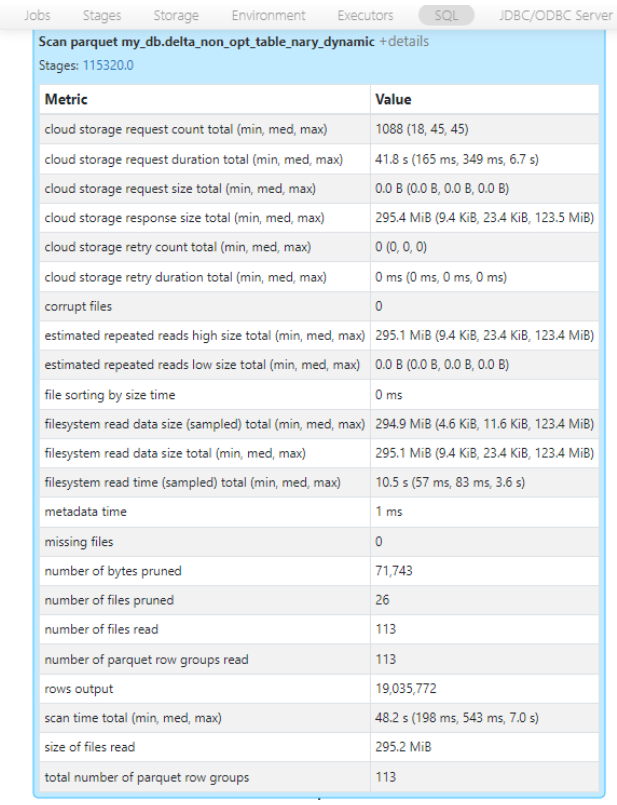
Και οι δύο πίνακες έχουν ακριβώς τα ίδια versions, με την διαφορά πως στον πίνακα delta_opt_table_nary_dynamic υπάρχει ένα παραπάνω version αυτό της εντολής OPTIMIZE που έχουμε εφαρμόσει.

Στην συνέχεια της υλοποίησης και οι δύο πίνακες θα κληθούν να απαντήσουν σε ένα αρκετά περίπλοκο query το οποίο περιέχει αρκετές aggregated functions σε συνδυασμό με μερικά φίλτρα.

7.4.3 Ανάλυση Αποτελεσμάτων

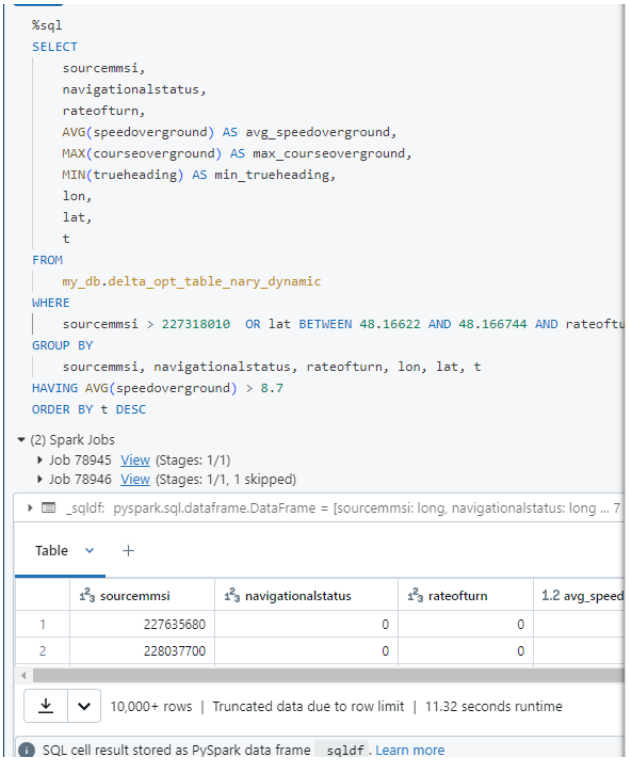
delta_non_opt_table_nary_dynamic





Metric	Value
cloud storage request count total (min, med, max)	1088 (18, 45, 45)
cloud storage request duration total (min, med, max)	41.8 s (165 ms, 349 ms, 6.7 s)
cloud storage request size total (min, med, max)	0.0 B (0.0 B, 0.0 B, 0.0 B)
cloud storage response size total (min, med, max)	295.4 MiB (9.4 KiB, 23.4 KiB, 123.5 MiB)
cloud storage retry count total (min, med, max)	0 (0, 0, 0)
cloud storage retry duration total (min, med, max)	0 ms (0 ms, 0 ms, 0 ms)
corrupt files	0
estimated repeated reads high size total (min, med, max)	295.1 MiB (9.4 KiB, 23.4 KiB, 123.4 MiB)
estimated repeated reads low size total (min, med, max)	0.0 B (0.0 B, 0.0 B, 0.0 B)
file sorting by size time	0 ms
filesystem read data size (sampled) total (min, med, max)	294.9 MiB (4.6 KiB, 11.6 KiB, 123.4 MiB)
filesystem read data size total (min, med, max)	295.1 MiB (9.4 KiB, 23.4 KiB, 123.4 MiB)
filesystem read time (sampled) total (min, med, max)	10.5 s (57 ms, 83 ms, 3.6 s)
metadata time	1 ms
missing files	0
number of bytes pruned	71,743
number of files pruned	26
number of files read	113
number of parquet row groups read	113
rows output	19,035,772
scan time total (min, med, max)	48.2 s (198 ms, 543 ms, 7.0 s)
size of files read	295.2 MiB
total number of parquet row groups	113

delta_opt_table_nary_dynamic



```
%sql
SELECT
    sourcemmsi,
    navigationalstatus,
    rateofturn,
    AVG(speedoverground) AS avg_speedoverground,
    MAX(courseoverground) AS max_courseoverground,
    MIN(trueheading) AS min_trueheading,
    lon,
    lat,
    t
FROM
    my_db.delta_opt_table_nary_dynamic
WHERE
    sourcemmsi > 227318010  OR lat BETWEEN 48.16622 AND 48.166744 AND rateoftu
GROUP BY
    sourcemmsi, navigationalstatus, rateofturn, lon, lat, t
HAVING AVG(speedoverground) > 8.7
ORDER BY t DESC
```

(2) Spark Jobs
Job 78945 View (Stages: 1/1)
Job 78946 View (Stages: 1/1, 1 skipped)

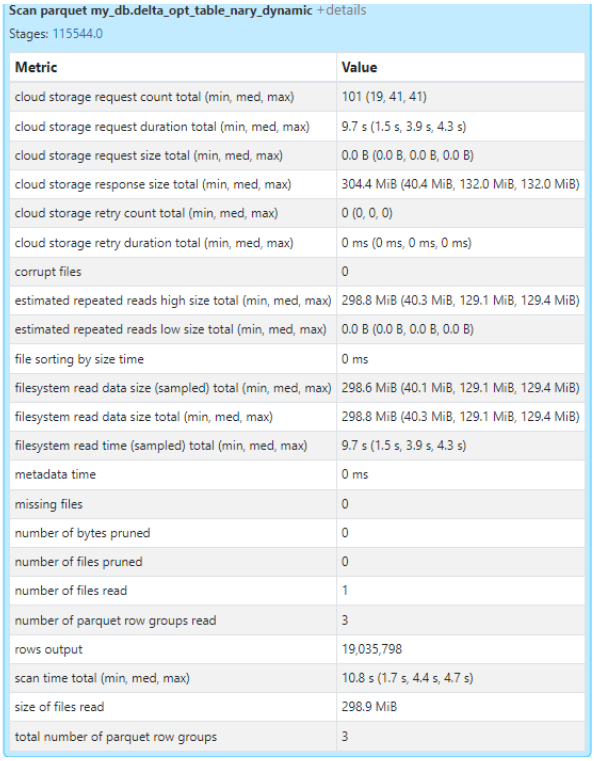
_sqldf: pyspark.sql.dataframe.DataFrame = [sourcemmsi: long, navigationalstatus: long ... 7

Table +

	sourcemmsi	navigationalstatus	rateofturn	1.2 avg_speed
1	227635680	0	0	
2	228037700	0	0	

10,000+ rows | Truncated data due to row limit | 11.32 seconds runtime

SQL cell result stored as PySpark data frame `sqldf`. Learn more



Scan parquet my_db.delta_opt_table_nary_dynamic +details

Stages: 115544.0

Metric	Value
cloud storage request count total (min, med, max)	101 (19, 41, 41)
cloud storage request duration total (min, med, max)	9.7 s (1.5 s, 3.9 s, 4.3 s)
cloud storage request size total (min, med, max)	0.0 B (0.0 B, 0.0 B, 0.0 B)
cloud storage response size total (min, med, max)	304.4 MiB (40.4 MiB, 132.0 MiB, 132.0 MiB)
cloud storage retry count total (min, med, max)	0 (0, 0, 0)
cloud storage retry duration total (min, med, max)	0 ms (0 ms, 0 ms, 0 ms)
corrupt files	0
estimated repeated reads high size total (min, med, max)	298.8 MiB (40.3 MiB, 129.1 MiB, 129.4 MiB)
estimated repeated reads low size total (min, med, max)	0.0 B (0.0 B, 0.0 B, 0.0 B)
file sorting by size time	0 ms
filesystem read data size (sampled) total (min, med, max)	298.6 MiB (40.1 MiB, 129.1 MiB, 129.4 MiB)
filesystem read data size total (min, med, max)	298.8 MiB (40.3 MiB, 129.1 MiB, 129.4 MiB)
filesystem read time (sampled) total (min, med, max)	9.7 s (1.5 s, 3.9 s, 4.3 s)
metadata time	0 ms
missing files	0
number of bytes pruned	0
number of files pruned	0
number of files read	1
number of parquet row groups read	3
rows output	19,035,798
scan time total (min, med, max)	10.8 s (1.7 s, 4.4 s, 4.7 s)
size of files read	298.9 MiB
total number of parquet row groups	3

Πιο αναλυτικά:

<u>Cloud storage request duration total:</u>

Είναι η μέτρηση του χρόνου που απαιτείται από το αίτημα που γίνεται από την μηχανή στον αποθηκευτικό χώρο αποθήκευσης των αρχείων του κάθε πίνακα (στην δική μας περίπτωση στο dbfs, το οποίο χώρος αποθήκευσης αρχείων του Databricks καθώς εκεί βρίσκονται αποθηκευμένα τα αρχεία μας) κατά την εκτέλεση του ερωτήματος.

- delta_non_opt_table_nary_dynamic: 41.8 s
- delta_opt_table_nary_dynamic: 9.7 s

Παρατηρείται πολύ μεγαλύτερη διαφορά μεταξύ των δύο πινάκων.

Numbers of files pruned:

Είναι ο αριθμός των αρχείων του πίνακα που η μηχανή δεν διαβάζει και τα «αγνοεί».

- delta_non_opt_table_nary_dynamic: 26

  Η μηχανή με την βοήθεια των metadata έχει προσπεράσει 26 αρχεία, καθώς γνώριζε πως δεν θα βρει εκεί την πληροφορία για το ερώτημα που της ανατέθηκε.

- delta_opt_table_nary_dynamic: 0

  Ο αριθμός 0 για τον πίνακα που έχει γίνει OPTIMIZED δηλώνει πως η μηχανή δεν έχει κόψει κανένα αρχείο, για να απαντήσει στο ερώτημα.

Numbers of files read:

- delta_non_opt_table_nary_dynamic: 113

Σε αυτή την περίπτωση η μηχανή διαβάζει 113 Parquet αρχεία για να απαντήσει.

- delta_opt_table_nary_dynamic: 1

Σε αντίθεση με τον OPTIMIZED πίνακα που διαβάζει μόνο ένα αρχείο. Αυτό φυσικά συμβαίνει γιατί με την τεχνική που εφαρμόστηκε, τα αρχεία που υπήρχαν έχουν συμπιεστεί σε ένα (δεν ήταν τόσο πολλά σε αριθμό ή μέγεθος για να συμπιεστούν σε παραπάνω από ένα αρχεία). Έχει μειωθεί σε πολύ μεγάλο βαθμό ο αριθμός των αρχείων που διαβάζεται.

Scan time total (min, med, max)

Μία εξίσου σημαντική μέτρηση είναι η συνολική ώρα που η μηχανή χρειάζεται για να διατρέξει όλα τα αρχεία.

- delta_non_opt_table_nary_dynamic: 48.2 s
- delta_opt_table_nary_dynamic: 10.8 s

Και σε αυτή την μέτρηση παρατηρείται αρκετή μείωση του χρόνου που χρειάστηκε η μηχανή για να σκανάρει τον OPTIMIZED Delta πίνακα.

7.5 Συμπέρασμα

Με την παραπάνω τεχνική που μπορεί να εφαρμοστεί μόνο στους Delta πίνακες, γίνεται αντιληπτό το γεγονός πως είναι εφικτή η βελτιστοποίηση των Delta, όπως αποδείχθηκε με την ανάλυση των στατιστικών των ερωτημάτων.

Ο συγκεκριμένος τρόπος βελτιστοποίησης λύνει τα χέρια στους μηχανικούς καθώς η συσσώρευση των Parquet αρχείων των Delta μπορεί να αυξάνεται με ραγδαίους ρυθμούς καθημερινά και ανεξέλεγκτα.

Άρα συμπυκνώνοντας πολλά αρχεία που μπορεί να έχει ένας πίνακας από τα πολλά Transactions που έχει υποστεί σε αρχεία συμπυκνωμένα αρχεία μέχρι του μεγέθους 1 GB. Κατά αυτό τον τρόπο λοιπόν μειώνεται δραματικά ο αριθμός των «ενεργών» αρχείων που θα πρέπει να διαβάσει η μηχανή ώστε να απαντήσει στα ερωτήματα που θα της τεθούν.

Στον παραπάνω πίνακα του πειράματος, έχει δημιουργηθεί ένα μόνο αρχείο μετά την εκτέλεση της OPTIMIZED εντολής καθώς το πλήθος των αρχείων που υπήρχαν, αλλά και το μέγεθος τους δεν επαρκούν ώστε να δημιουργήσουν αρχείο παραπάνω του 1GB, ώστε η μηχανή να δημιουργήσει παραπάνω συμπιεσμένα τέτοια αρχεία.

Κεφάλαιο 8: Συμπέρασμα

Τα συμπεράσματα της διπλωματικής εργασίας αναδεικνύουν τις σημαντικές βελτιώσεις στην απόδοση των ερωτημάτων όταν χρησιμοποιούνται ταξινομημένα αρχεία Parquet, καθώς και τη βελτιστοποίηση των Delta tables.

8.1 Parquet- CSV

Η χρήση ταξινομημένων αρχείων Parquet προσφέρει σημαντικά πλεονεκτήματα στην απόδοση των ερωτημάτων σε σύγκριση με τα αρχεία CSV και τα μη ταξινομημένα αρχεία Parquet, όπως έχει αναδειχθεί από τα πειράματα στο πρώτο μέρος της διπλωματικής.

- Τα ταξινομημένα αρχεία Parquet βελτιώνουν σημαντικά την απόδοση των ερωτημάτων που επιστρέφουν λίγες εγγραφές και απαιτούν την ανάγνωση μικρού και στοχευμένου μέρους του αρχείου. Αυτό συμβαίνει επειδή τα δεδομένα είναι οργανωμένα-ταξινομημένα, σε κάθε row group επιτρέποντας πιο αποτελεσματική και όχι την ανούσια αναζήτηση μέσα στο αρχείο, για τα περίπλοκα ερωτήματα.
- Ο μειωμένος χρόνος ερωτήματος και η μειωμένη χρήση μνήμης στα ταξινομημένα αρχεία Parquet δείχνουν τα πλεονεκτήματα της ταξινόμησης δεδομένων για συγκεκριμένους τύπους ερωτημάτων, ιδιαίτερα εκείνων που αφορούν πολύπλοκων συνθηκών.

Συνολικά, η χρήση των ταξινομημένων αρχείων Parquet προσφέρει σημαντικές βελτιώσεις απόδοσης για περίπλοκα και απαιτητικά ερωτήματα (το ίδιο δεν συμβαίνει για ερωτήματα που επιστρέφουν μεγάλο μέρος του αρχείου, όπως έχει αποδειχθεί παραπάνω) καθώς επιτρέπουν την αποτελεσματική διαχείριση των δεδομένων και την γρήγορη ανάκτηση των απαιτούμενων πληροφοριών.

8.2 Delta Tables

Η βελτιστοποίηση των Delta Tables αποτελεί μια κρίσιμη τεχνική για την αποδοτική διαχείριση δεδομένων, ιδιαίτερα σε περιβάλλοντα όπου η συσσώρευση δεδομένων είναι συνεχής και αυξανόμενη. Παρόλο αυτά υπάρχουν τεχνικές (όπως αναφέρθηκαν παραπάνω) για εκ νέου δυνατότητα βελτιστοποίησης τους. Η συγκεκριμένη τεχνική ωστόσο (του Z-Ordering που επιλέχθηκε για να εφαρμοστεί) εστιάζει στην συμπύκνωση-μείωση των πολλών σε αριθμό ενεργών αρχείων Parquet, που πρέπει να διαβάσει η μηχανή για να απαντήσει σε κάθε ερώτημα. Γίνεται αντιληπτό πως κάθε πίνακας καθημερινά υπόκεινται σε πολλές συναλλαγές πράγμα που σημαίνει πως προκύπτει η δημιουργία πολλών αρχείων (Parquet) στην τοποθεσία αποθήκευσης του, άρα μια τέτοια τεχνική καθίστανται αναγκαία για κάθε Delta πίνακα.

Κανείς μπορεί να αναλογιστεί πως η χρήση-μέθοδος αποθήκευσης των δεδομένων μέσω των Delta Tables γίνεται όταν επρόκειτο για μεγάλο όγκο δεδομένων που υποβάλλεται καθημερινά σε συναλλαγές και φυσικά σε συνδυασμό με μεθόδους βελτιστοποίησης μπορεί να υπάρχουν περισσότερα οφέλη όπως:

- **Βελτιωμένη Απόδοση Συστήματος:** Η μείωση του αριθμού των ενεργών αρχείων οδηγεί σε ταχύτερη εκτέλεση ερωτημάτων και καλύτερη απόδοση του συστήματος.
- **Απλοποιημένη Διαχείριση Δεδομένων:** Η συγκέντρωση των δεδομένων σε λιγότερα, μεγαλύτερα αρχεία διευκολύνει τη διαχείριση και τη συντήρηση των δεδομένων.
- **Μείωση Κόστους:** Η βελτιστοποιημένη αποθήκευση δεδομένων μπορεί να συμβάλλει στη μείωση του κόστους αποθήκευσης και επεξεργασίας.
- **Αξιοπιστία και Συνέπεια:** Η βελτιστοποίηση διασφαλίζει ότι τα δεδομένα είναι συνεκτικά και αξιόπιστα, μειώνοντας τις πιθανότητες σφαλμάτων και καθυστερήσεων.

Συμπερασματικά, η βελτιστοποίηση των Delta Tables αποτελεί ένα σημαντικό εργαλείο για την επίτευξη αποδοτικότερης και πιο αξιόπιστης διαχείρισης δεδομένων, προσφέροντας ουσιαστικά πλεονεκτήματα τόσο για τις επιχειρήσεις όσο και για τους μηχανικούς.

Βιβλιογραφία

1) https://blogs.perficient.com/2022/01/07/8-ways-to-data-scientists-can-optimize-their-parquet-queries/
2) https://www.influxdata.com/blog/querying-parquet-millisecond-latency/
3) https://www.upsolver.com/blog/apache-parquet-why-use
4) https://learncsdesigns.medium.com/understanding-apache-parquet-d722645cfe74
5) https://www.dremio.com/blog/tuning-parquet/
6) https://data-mozart.com/parquet-file-format-everything-you-need-to-know/
7) https://thenewstack.io/an-introduction-to-apache-parquet/
8) https://www.databricks.com/glossary/what-is-parquet
9) https://www.chaosgenius.io/blog/delta-table/
10) https://delta.io/blog/delta-lake-vs-parquet-comparison/
11) https://www.boltic.io/blog/databricks-delta
12) https://docs.delta.io/latest/delta-intro.html
13) https://medium.com/@harshavardhan.achyuta/understanding-managed-and-external-tables-in-apache-spark-6e359dd2e5db
14) https://docs.databricks.com/en/introduction/index.html
15) https://chetnachaudhari.medium.com/delta-tables-optimization-techniques-8a1b2df7d4ef