Συστήματα Διαχείρισης Βάσεων Δεδομένων

Τεχνικη αναφορα εργασιας

Μαρια δημητρουλη π21028, γεωργιος νικολαïδης π21115

2024

Περιεχόμενα

[Σύντομη εξήγηση των queries. 2](#_Toc158062193)

[Query 1 2](#_Toc158062194)

[Query 2 2](#_Toc158062195)

[Query 3 2](#_Toc158062196)

[Query 4 2](#_Toc158062197)

[Query 5 3](#_Toc158062198)

[Query 5A 3](#_Toc158062199)

[Query 5B 3](#_Toc158062200)

[Ερώτημα 1 4](#_Toc158062201)

[Κατασκευή πινάκων και εισαγωγή δεδομένων CSV 4](#_Toc158062202)

[Πλάνα εκτέλεσης 5](#_Toc158062203)

[Χρόνοι εκτέλεσης 8](#_Toc158062204)

[Ερώτημα 2 8](#_Toc158062205)

[Αύξηση της τιμής της παραμέτρου shared\_buffers 8](#_Toc158062206)

[Πλάνα Εκτέλεσης 8](#_Toc158062207)

[Χρόνοι Εκτέλεσης 8](#_Toc158062208)

[Ερώτημα 3 9](#_Toc158062209)

[Αύξηση της υπολογιστικής ισχύς του PostgreSQL μέσω max\_parallel\_workers\_per\_gather 9](#_Toc158062210)

[Πλάνα εκτέλεσης 9](#_Toc158062211)

[Χρόνοι εκτέλεσης 11](#_Toc158062212)

[Ερώτημα 4 12](#_Toc158062213)

[Επιλογή και υλοποίηση ευρετηρίων 12](#_Toc158062214)

[Πλάνα εκτέλεσης 12](#_Toc158062215)

[Χρόνοι εκτέλεσης 15](#_Toc158062216)

[Ερώτημα 5 15](#_Toc158062217)

[Επιλογή μεθόδου και τρόπου PARTITION 15](#_Toc158062218)

[Πλάνα εκτέλεσης 16](#_Toc158062219)

[Χρόνοι Εκτέλεσης 22](#_Toc158062220)

# Σύντομη εξήγηση των **queries.**

## Query 1

SELECT DATE(t) AS date, lon, lat, COUNT(\*) AS count

FROM positions

GROUP BY date, lon, lat

ORDER BY count DESC;

Εδώ κάλλιστα θα μπορούσαμε να αφήσουμε τα πεδία lon, lat και να κάνουμε ένα απλό GROUP BY με την ημερομηνία μόνο. Παρόλα αυτά εμείς έτσι ερμηνεύσαμε το την εκφώνηση και έτσι το γράψαμε. Αν τρέξουμε το query βλέπουμε ότι μας παρέχει χρήσιμες πληροφορίες για το που συχνάζουν (lon, lat) τα πλοία ανά ημέρα.

## Query 2

SELECT type, COUNT(\*)

FROM Vessels

WHERE flag='Greece'

GROUP BY type

Πολύ απλό query που δεν χρειάζεται περαιτέρω εξήγηση

## Query 3

WITH Vessels\_over\_30 AS (

    SELECT type, vessel\_id

    FROM Positions AS p

    INNER JOIN Vessels AS v ON p.vessel\_id=v.id

    WHERE speed>30

    GROUP BY type, vessel\_id

    )

SELECT type, COUNT(\*)

FROM Vessels\_over\_30

GROUP BY type

Εδώ αρχικά εκτελούμε το subquery με όνομα **Vessels\_over\_30** το οποίο επιστρέφει τα πλοία που κάποια στιγμή ανέπτυξαν ταχύτητα >30 κόμβων καθώς και τον τύπο τους. Το GROUP BY στο WITH υπάρχει έτσι ώστε κάθε πλοίο να επιστρέφεται ΜΙΑ φορά (ανάλογο με το SELECT DISTINCT). Ύστερα, μετράμε τα πλοία ανά τύπο από το Vessels\_over\_30

## Query 4

SELECT date(t) AS date, COUNT(\*)

FROM Positions AS p

INNER JOIN Vessels AS v ON p.vessel\_id=v.id

INNER JOIN VesselTypes AS vt ON v.type=vt.code

WHERE description LIKE 'Passenger%' AND t >= '2019-08-14 00:00:00' AND t <= '2019-08-18 23:59:59'

GROUP BY date

Αρχικά κάνουμε JOIN όλους τους πίνακες μεταξύ τους καθώς χρειαζόμαστε δεδομένα όπως το χρόνο στίγματος (Positions.t) και την περιγραφή του τύπου του πλοίου (VesselTypes.description). Μετά φιλτράρουμε τις πλειάδες και συγκρατούμε αυτές που έχουν καταγραφεί στην περίοδο 14/08/2019 – 18/08/2019. Μετά, γίνεται pattern matching και τέλος μετράμε τις πλειάδες ανά ημερολογιακή ημέρα.

## Query 5

Εδώ φτιάξαμε δυο ξεχωριστά queries που απαντούν στα ερωτήματα

«

Ποια πλοία τύπου cargo ήταν ‘αγκυροβολημένα’ (ταχύτητα μηδέν) κάποια στιγμή μέσα στην περίοδο 15/08/2019 - 18/08/2019;

* 1. »

και

«

* 1. Ποια για ολόκληρη την περίοδο 12/08/2019 - 19/08/2019;
  2. »

Καθώς θα μπορούσαμε να κάνουμε ένα query (με την ένωση (ΟR) των συνθηκών) αποφασίσαμε να τα χωρίσουμε σε δύο για να είναι πιο ξεκάθαρο ποια πλοία ανήκουν στο πρώτο ερώτημα και ποια στο δεύτερο.

### Query 5A

SELECT DISTINCT vessel\_id, description

FROM Positions AS p

INNER JOIN Vessels AS v ON p.vessel\_id=v.id

INNER JOIN VesselTypes AS vt ON v.type=vt.code

WHERE description LIKE 'Cargo%' AND speed = 0 AND t >= '2019-08-15 00:00:00' AND t <= '2019-08-18 23:59:59'

Αυτό το query απαντάει στο πρώτο ερώτημα, η σύνταξη του όσον αφορά τα JOIN και το φιλτράρισμα είναι παρόμοια με αυτή του query 4. Τέλος, επιστρέφουμε μια μοναδική πλειάδα για κάθε πλοίο που τηρεί τις συνθήκες

### Query 5B

WITH sum\_speed AS (

    SELECT DISTINCT vessel\_id, sum(speed)

    FROM Positions

    WHERE t >= '2019-08-12 00:00:00' AND t <= '2019-08-19 23:59:59'

    GROUP BY vessel\_id

    HAVING sum(speed) <= 1.5

)

SELECT vessel\_id, description

FROM sum\_speed AS s

INNER JOIN Vessels AS v ON s.vessel\_id = v.id

INNER JOIN VesselTypes AS vt ON v.type = vt.code

WHERE description LIKE 'Cargo%'

Αρχικά εκτελούμε το subquery **sum\_speed** για να πάρουμε τα πλοία στο χρονικό διάστημα 12/08/2019 – 19/08/2019 τα οποία ήταν αγκυροβολημένα κατά ΟΛΟΚΛΗΡΗ τη περίοδο αυτή (το καταφέρνουμε αυτό με το HAVING sum(speed) <= 1.5). Έπειτα, από το sum\_speed, φιλτράρουμε τα πλοία που δεν είναι τύπου ‘Cargo’ και μένουμε μόνομε τα φορτηγά (cargo) πλοία.

# Ερώτημα 1

## Κατασκευή πινάκων και εισαγωγή δεδομένων CSV

Αρχικά, φτιάχνουμε το schema (τους πίνακες) και ύστερα εισάγουμε τα δεδομένα από τα αρχεία CSV.

/\*

    1. Copy CSV contents to tables

    2. Add constraints (PK, FK, ...)

\*/

CREATE TABLE VesselTypes (

    code INTEGER,

    description TEXT

);

COPY VesselTypes FROM 'C:/Users/Public/dataset/VesselTypes.csv' WITH DELIMITER AS ',' CSV HEADER;

CREATE TABLE Vessels (

    id VARCHAR(64),

    type INTEGER,

    flag VARCHAR(32)

);

COPY Vessels FROM 'C:/Users/Public/dataset/Vessels.csv' WITH DELIMITER AS ',' CSV HEADER;

CREATE TABLE Positions (

    id INTEGER,

    vessel\_id VARCHAR(64),

    t TIMESTAMP,

    lon DOUBLE PRECISION,

    lat DOUBLE PRECISION,

    heading INTEGER,

    course DOUBLE PRECISION,

    speed DOUBLE PRECISION

);

COPY Positions FROM 'C:/Users/Public/dataset/Positions.csv' WITH DELIMITER AS ',' CSV HEADER;

/\* CONSTRAINTS \*/

ALTER TABLE VesselTypes ADD PRIMARY KEY (code);

ALTER TABLE Vessels ADD PRIMARY KEY (id);

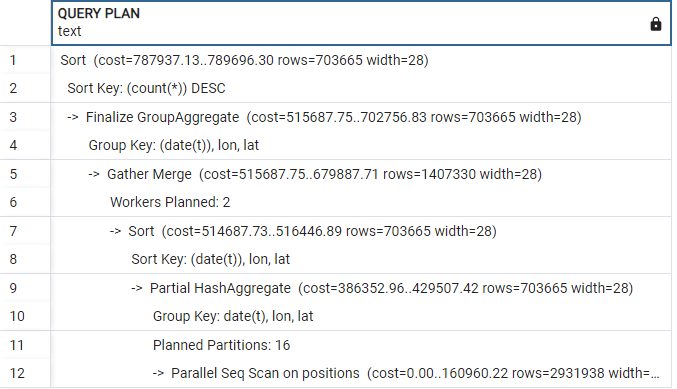
ALTER TABLE Vessels ADD CONSTRAINT vesfk FOREIGN KEY (type) REFERENCES VesselTypes(code);

ALTER TABLE Positions ADD PRIMARY KEY (id);

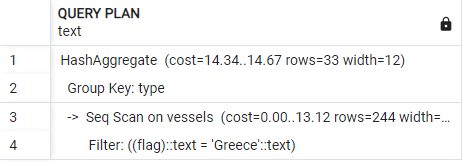
ALTER TABLE Positions ADD CONSTRAINT posfk FOREIGN KEY (vessel\_id) REFERENCES Vessels(id);

Ύστερα, κάνουμε **VACUUM FULL** και συνεχίζουμε στην χρονομέτρηση και στα πλάνα εκτέλεσης.

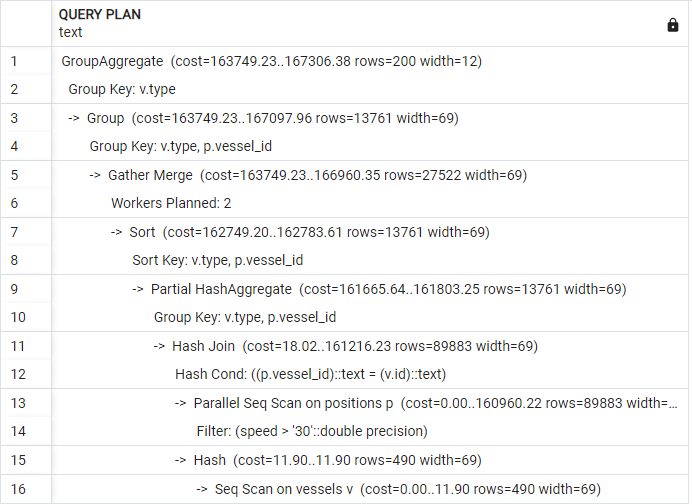
## Πλάνα εκτέλεσης



Εικόνα i.1: Πλάνο εκτέλεσης για το query 1 στο ερώτημα 1



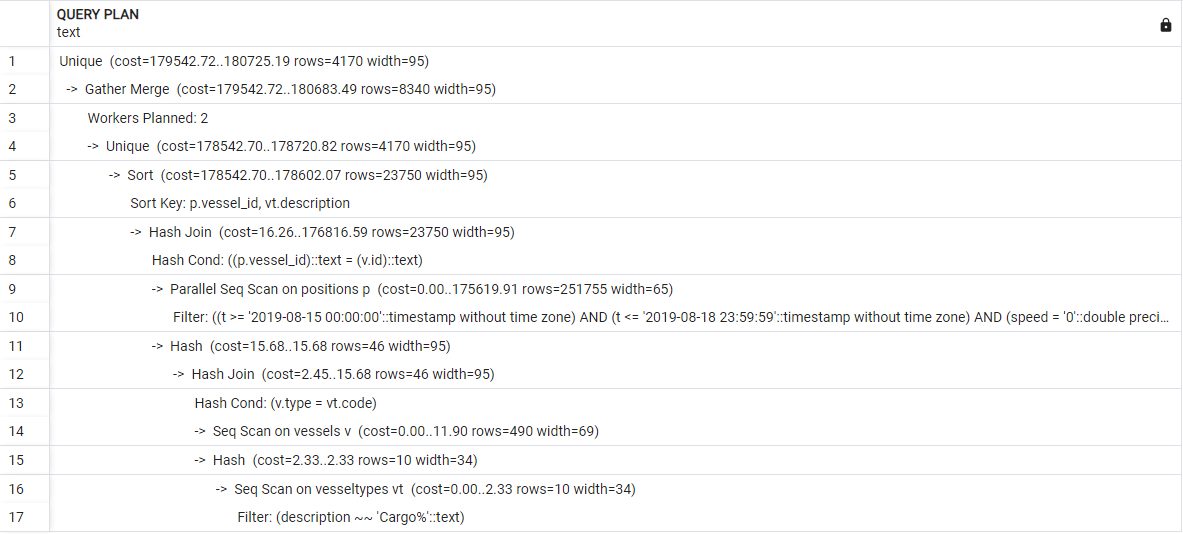
Εικόνα i.2: Πλάνο εκτέλεσης για το query 2 στο ερώτημα 1



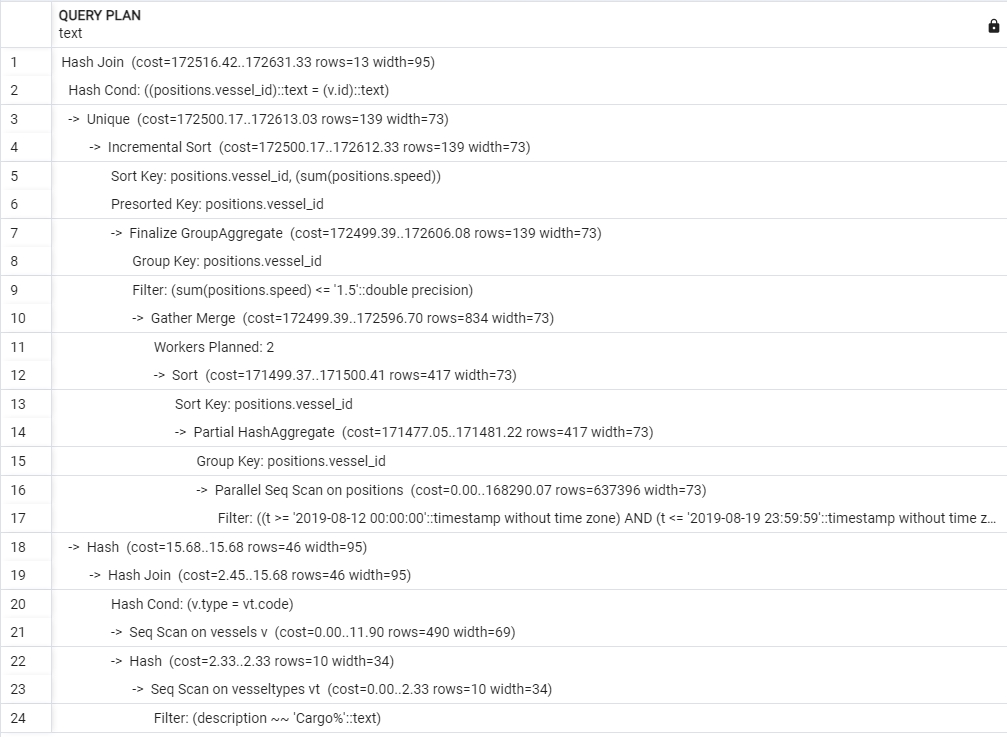
Εικόνα i.3: Πλάνο εκτέλεσης query 3 στο ερώτημα 1



Εικόνα i.4: Πλάνο εκτέλεσης query 4 στο ερώτημα 1



Εικόνα i.5: Πλάνο εκτέλεσης query 5A για το ερώτημα 1



Εικόνα i.6: Πλάνο εκτέλεσης query 5B για το ερώτημα 1

## Χρόνοι εκτέλεσης

|  |  |  |  |
| --- | --- | --- | --- |
| **Query** | **Χρόνος (sec)** | |  |
|  | Mary | George | Average |
| 1 | 11,078 | 7,333 | 9,2055 |
| 2 | 0,101 | 0,06 | 0,0805 |
| 3 | 1,096 | 0,613 | 0,8545 |
| 4 | 1,038 | 0,595 | 0,8165 |
| 5A | 1,116 | 0,613 | 0,8645 |
| 5B | 1,201 | 0,64 | 0,9205 |

# Ερώτημα 2

## Αύξηση της τιμής της παραμέτρου shared\_buffers

Με την παρακάτω εντολή και την επανεκκίνηση του PostgreSQL server, αυξάνουμε τα shared\_buffers από 128MB (το ξέρουμε επειδή κάναμε SHOW shared\_buffers;) σε 1GB.

ALTER SYSTEM SET shared\_buffers = '1GB';

Ρυθμίσαμε τα shared\_buffers να έχουν χώρο 1GB επειδή το dataset ολόκληρο περιλαμβάνει ~800MB χώρου στον δίσκο. Έτσι, μπορεί ολόκληρο να χωρέσει στα buffers.

## Πλάνα Εκτέλεσης

Δεν άλλαξε τίποτα από τα πλάνα εκτέλεσης του 1ου ερωτήματος, άρα παραλείπουμε τα screenshots.

## Χρόνοι Εκτέλεσης

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query** | **Χρόνος (sec)** | |  | **Βελτίωση** |
|  | Mary | George | Average |  |
| 1 | 9,534 | 7,175 | 8,3545 | -9,24% |
| 2 | 0,113 | 0,056 | 0,0845 | 4,97% |
| 3 | 1,119 | 0,654 | 0,8865 | 3,74% |
| 4 | 1,03 | 0,631 | 0,8305 | 1,71% |
| 5A | 1,059 | 0,643 | 0,851 | -1,56% |
| 5B | 1,119 | 0,7 | 0,9095 | -1,20% |

Παρατηρούμε ότι με εξαίρεση το query 1, δεν υπάρχει βελτίωση από το προηγούμενο ερώτημα. Αρχικά, στο ερώτημα 1, ο PostgreSQL σκανάρει ΟΛΟ τον πίνακα Positions (μέγεθος ~800MB), αυτό εξηγεί την ~10% βελτίωση στο χρόνο εκτέλεσης του ερωτήματος 1. Πριν, με shared\_buffers = ‘128MB’ δεν ήταν δυνατό να χωρέσει ολόκληρο τον πίνακα Positions στη μνήμη, άρα θα υπήρχε ανάγκη για περισσότερα I/O calls. Για τα υπόλοιπα ερωτήματα όμως, από τα πλάνα εκτέλεσης βλέπουμε ότι τα rows που θα επιλεχθούν χωράνε σε buffer <128MB, άρα δεν υπάρχει διαφορά στο χρόνο εκτέλεσης.

# Ερώτημα 3

## Αύξηση της υπολογιστικής ισχύς του PostgreSQL μέσω max\_parallel\_workers\_per\_gather

Εκτός από την αλλαγή της παραμέτρου **max\_parallel\_workers\_per\_gather**, θα αλλάξουμε και τις παραμέτρους **max\_worker\_processes** και **max\_parallel\_workers** επειδή σύμφωνα με το [documentation της PostgreSQL](https://postgresqlco.nf/doc/en/param/max_parallel_workers_per_gather/), πρέπει να αυξήσουμε και τις τιμές των δύο επιπλέον παραμέτρων για να λειτουργούν περισσότεροι από 8 parallel\_workers\_per\_gather. Εμείς επιλέξαμε την τιμή 64 και για τις τρείς παραμέτρους

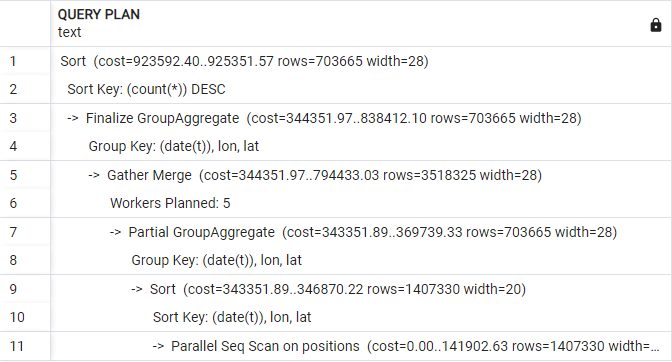
ALTER SYSTEM SET max\_worker\_processes = 64;

/\* Restart server here\*/

SET max\_parallel\_workers = 64;

SET max\_parallel\_workers\_per\_gather = 64;

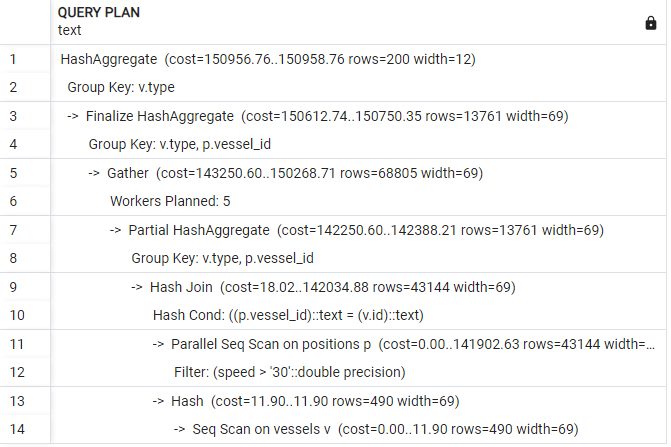
## Πλάνα εκτέλεσης



Εικόνα iii.0.1: Πλάνο εκτέλεσης query 1 για το ερώτημα 3

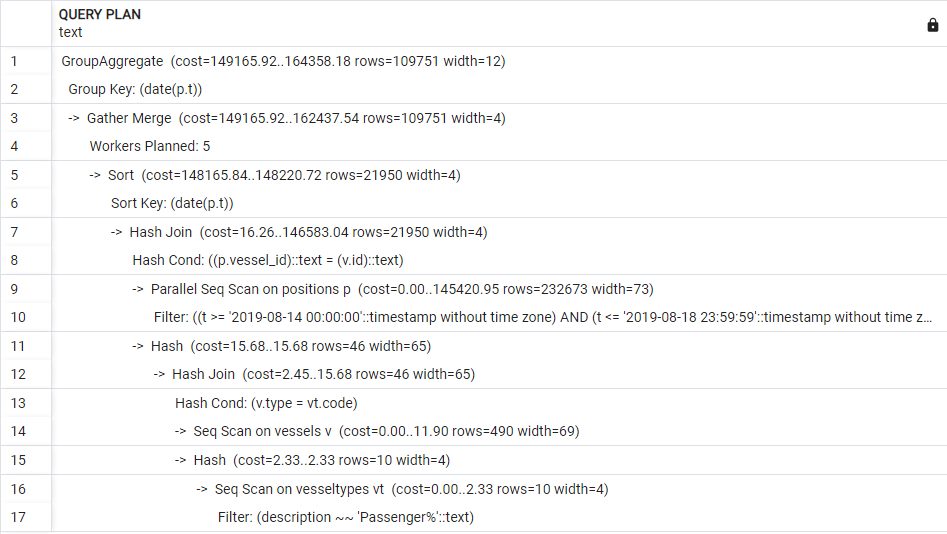
Σε σύγκριση με το ερώτημα 1, δεν έχουμε Planned Partitions και οι λειτουργίες Sort και Aggregate έχουν αλλάξει θέσεις. Το Partial HashAggregate έγινε Partial GroupAggregate και τέλος οι **Planned Workers** για το Gather Merge αυξήθηκαν από 2 σε 5, το οποίο είναι συνέπεια του ερωτήματος 3.

**Στο query 2 δεν υπάρχει αλλαγή του πλάνο εκτέλεσης.**

****

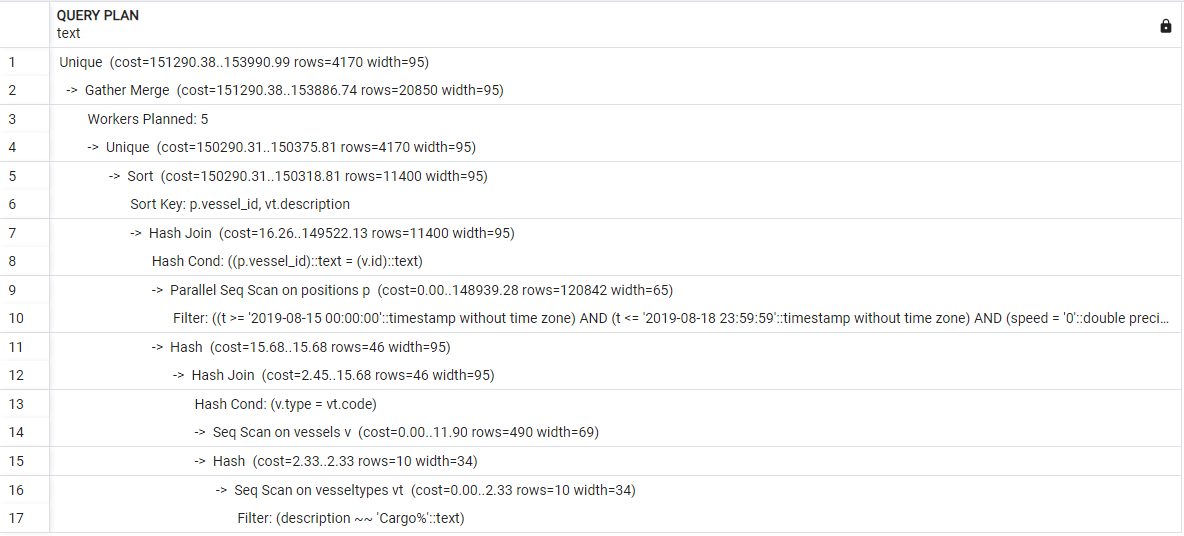
Εικόνα iii.0.2: Πλάνο εκτέλεσης query 3 για το ερώτημα 3

Παρατηρούμε πάλι την αύξηση των Planned Workers από 2 σε 5. Εδώ παραλείπεται το Sort (γραμμές 7 & 8 στο πλάνο του ερωτήματος 1) και το GroupAggregate γίνεται HashAggregate.



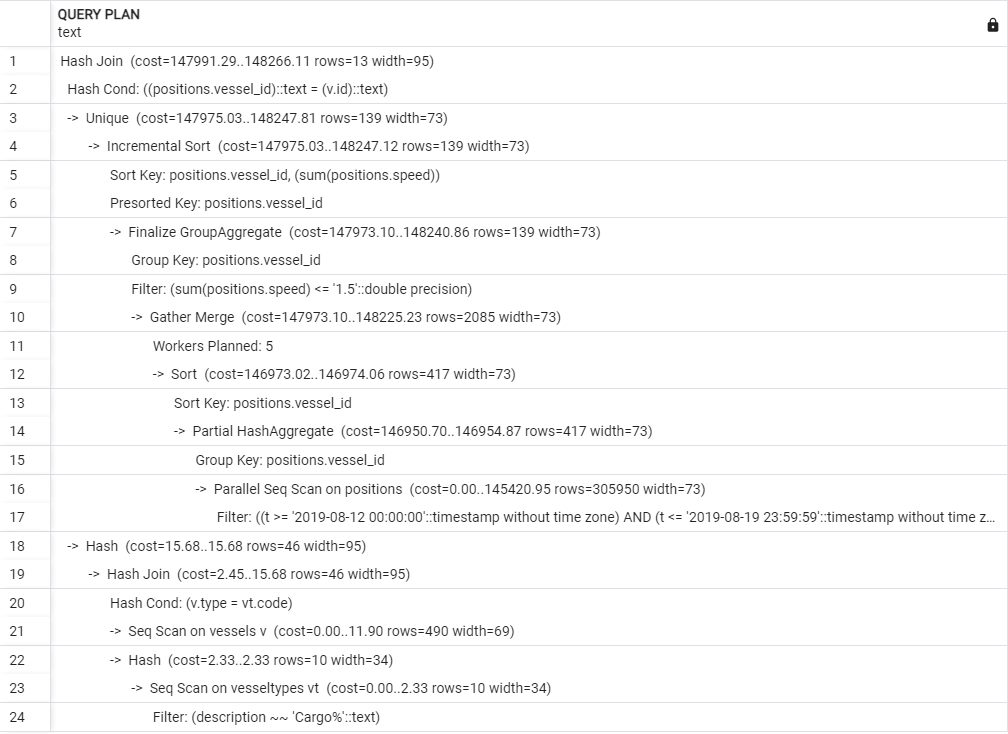
Εικόνα iii.0.3: Πλάνο εκτέλεσης query 4 για το ερώτημα 3

Αύξηση Planned Workers, παράληψη Partial GroupAggregate (γραμμές 5 & 6 στο πλάνο του ερωτήματος 1).



Εικόνα iii.0.4: Πλάνο εκτέλεσης query 5A για το ερώτημα 3

Ίδια δομή πλάνου με το ερώτημα 1, αύξηση Planned Workers.



Εικόνα iii.0.5: Πλάνο εκτέλεσης query 5B για το ερώτημα 3

Ίδια δομή πλάνου, αύξηση Planned Workers.

## Χρόνοι εκτέλεσης

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query** | **Xρόνος (sec)** |  |  | **Βελτίωση** |
|  | Mary | George | Average |  |
| 1 | 6,642 | 6,186 | 6,414 | -23,23% |
| 2 | 0,111 | 0,055 | 0,083 | -1,78% |
| 3 | 1,007 | 0,569 | 0,788 | -11,11% |
| 4 | 1,095 | 0,604 | 0,8495 | 2,29% |
| 5A | 0,978 | 0,557 | 0,7675 | -9,81% |
| 5B | 1,022 | 0,56 | 0,791 | -13,03% |

Παρατηρούμε μεγάλες βελτιώσεις στους χρόνους εκτέλεσης των queries, αυτό είναι χάρη κυρίως της αύξησης των Planned Workers per Gather. Η ουσιαστική μηδενική βελτίωση στο query 2 εξηγείται επειδή το πλάνο εκτέλεσης ΔΕΝ άλλαξε. Η χειροτέρευση του χρόνου στο query 4 μπορεί να ερμηνευτεί επίσης ως μηδενική βελτίωση. [Σε σύγκριση με τους χρόνους του ερωτήματος 2](#_Χρόνοι_Εκτέλεσης), ο χρόνος στον υπολογιστή του Τζώρτζη **βελτιώνεται** αλλά ο χρόνος στον υπολογιστή της Μαρίας **αυξάνεται** κατά μεγαλύτερο ποσοστό.

# Ερώτημα 4

## Επιλογή και υλοποίηση ευρετηρίων

Με βάση τα queries, δοκιμάσαμε αρκετά ευρετήρια αλλά καταλήξαμε στο να χρησιμοποιήσουμε μόνο 2.

Η επιλογή των ευρετηρίων έγινε με τα εξής κριτήρια:

1. Έχει ο πίνακας του πεδίου που θέλουμε να κάνουμε ευρετήριο **πολλές εγγραφές;**
2. Το πεδίο **χρησιμοποιείται στα queries που έχουμε φτιάξει**; Πιο συγκεκριμένα, χρησιμοποιείται σε **WHERE;**
3. Παρατηρούμε στην πράξη **βελτίωση στο χρόνο εκτέλεσης**;

Από το κριτήριο (1), μπορούμε να παραλείψουμε τους πίνακες Vessels και VesselTypes επειδή έχουν μικρό αριθμό εγγραφών και η δημιουργία ευρετηρίων στα πεδία αυτών των πινάκων δεν θα βελτιώσει το χρόνο εκτέλεσης. Αρά, περιοριζόμαστε στο πίνακα Positions για τα ευρετήρια.

Από το κριτήριο (2), παρατηρούμε ότι χρησιμοποιούμε τα πεδία **Positions.speed** και **Positions.t** σε WHERE.

Ας κατασκευάσουμε ευρετήρια για αυτά τα δύο πεδία. Θα χρησιμοποιήσουμε **B+Tree** και για τα δύο ευρετήρια επειδή κάνουμε ερωτήσεις διαστήματος.

CREATE INDEX pos\_speed\_index ON Positions USING btree (speed);

CREATE INDEX pos\_t\_index ON Positions USING btree (t);

## Πλάνα εκτέλεσης

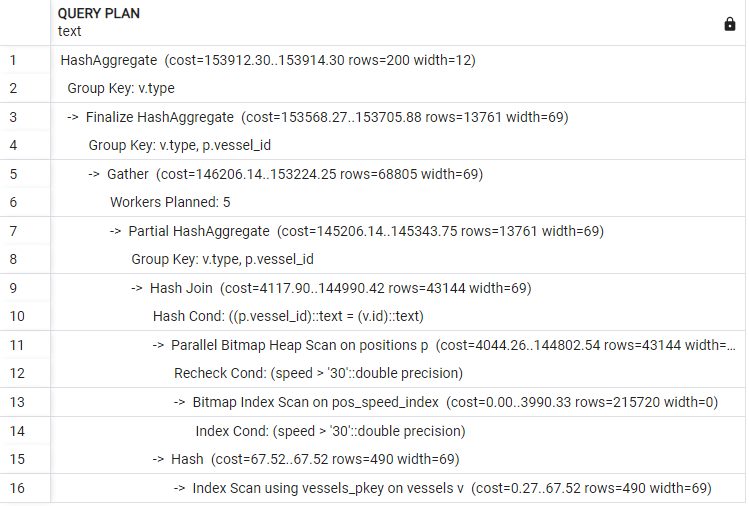
**Για τα query 1, 2 δεν υπάρχει αλλαγή στο πλάνο εκτέλεσης από το ερώτημα 3.**

Το παραπάνω προφανώς συμβαίνει επειδή δεν γίνεται χρήση των ευρετηρίων στα πρώτα δυο query.

Για τα επόμενα query εκτελούμε την εντολή

SET enable\_seqscan = 'OFF';

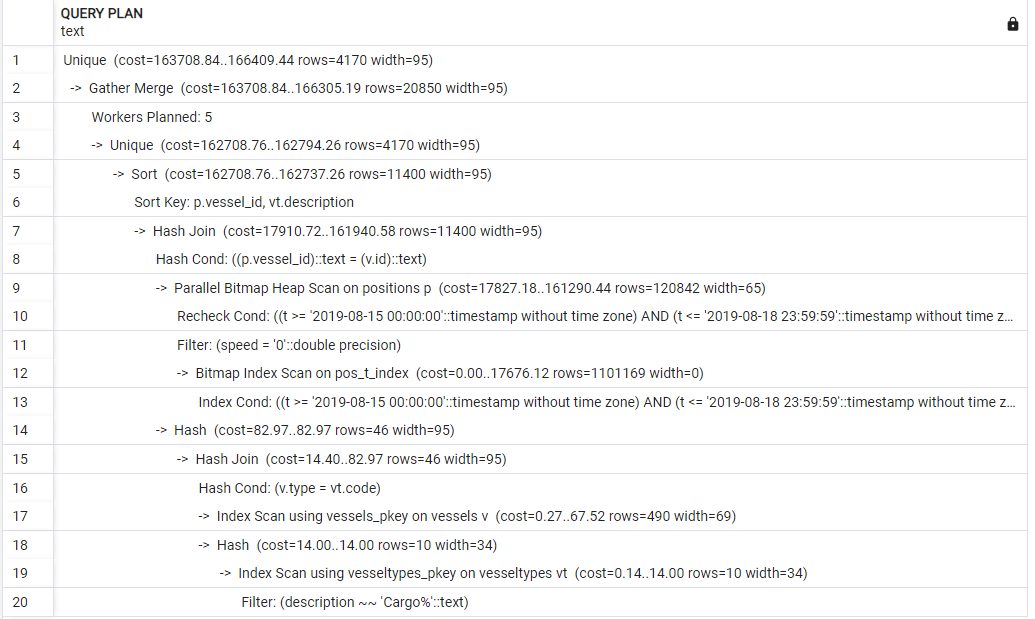
επειδή για κάποιο λόγο ο PostgreSQL δεν χρησιμοποιεί τα ευρετήρια από μόνος του, παρόλο την βελτίωση χρόνου που θα δειχθεί παρακάτω.



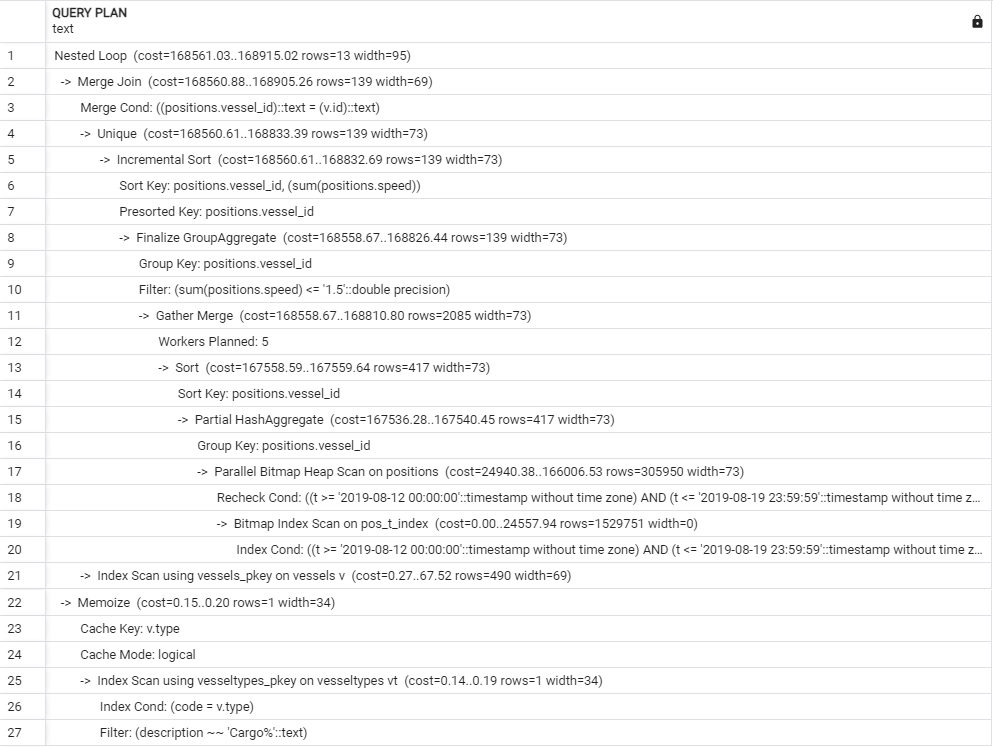
Εικόνα 0.1: Πλάνο εκτέλεσης query 3 για το ερώτημα 4



Εικόνα 0.2: Πλάνο εκτέλεσης query 4 για το ερώτημα 4



Εικόνα 0.3: Πλάνο εκτέλεσης query 5A για το ερώτημα 4



Εικόνα 0.4: Πλάνο εκτέλεσης query 5B για το ερώτημα 4

## Χρόνοι εκτέλεσης

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query** | **Xρόνος (sec)** |  |  | **Βελτίωση** |
|  | Mary | George | Average |  |
| 1 | 8,134 | 5,756 | 6,945 | 8,28% |
| 2 | 0,147 | 0,068 | 0,1075 | 29,52% |
| 3 | 0,317 | 0,238 | 0,2775 | -64,78% |
| 4 | 0,436 | 0,39 | 0,413 | -51,38% |
| 5A | 0,325 | 0,228 | 0,2765 | -63,97% |
| 5B | 0,475 | 0,39 | 0,4325 | -45,32% |

Στα queries που χρησιμοποιήθηκαν τα ευρετήρια παρατηρούμε την μείωση του χρόνου εκτέλεσης ΔΙΑ 2, αυτή η μείωση είναι σπουδαία. Εξίσου σπουδαία είναι η αύξηση του χρόνου εκτέλεσης. Αυτή δεν οφείλεται όμως στην ύπαρξη των ευρετηρίων καθώς τα πλάνα εκτέλεσης των queries 1, 2 είναι ίδια με αυτά του ερωτήματος 3.

# Ερώτημα 5

## Επιλογή μεθόδου και τρόπου PARTITION

Αρχικά, για τη μέθοδο διαμέρισης επιλέξαμε την **δηλωτική** (**declarative partitioning**) λόγω εύκολης χρήσης και πρακτικότητας συγκριτικά με την διαμέριση με κληρονομικότητα (table inheritance).

Για τον τρόπο διαμέρισης (RANGE, HASH, RANDOM, …), η αρχική μας επιλογή ήταν με **RANGE** στο πεδίο Positions.t. Η επιλογή αυτή στηρίχθηκε στη λογική διαμέρισης του πίνακα Positions ανά εβδομάδα. Στη πράξη όμως δεν είδαμε καμία βελτίωση. Στη συνέχεια, πειραματιστήκαμε με τον τρόπο **HASH** στο πεδίο Positions.id, βασιζόμενοι σε αυτά που κάναμε στο Ερώτημα 3. Το σκεπτικό ήταν πως αν διαχωρίσουμε το πίνακα Positions σε ίσους σε πλήθος εγγραφών υποπίνακες, τότε η PostgreSQL θα μπορέσει να σαρώσει παράλληλα τους υποπίνακες (εφόσον από το Ερώτημα 3 οι μέγιστοι Parallel Workers αυξήθηκαν) και να ολοκληρώσει πιο γρήγορα.

Τέλος, για κάθε υποπίνακα δημιουργήσαμε ευρετήρια τύπου B+Tree για τα πεδία t και speed, όπως κάναμε και στο Ερώτημα 4.

/\* Δημιουργία πίνακα διαμέρισης \*/

CREATE TABLE Positions\_Hash\_Partition (

    id INTEGER PRIMARY KEY,

    vessel\_id VARCHAR(64),

    t TIMESTAMP,

    lon DOUBLE PRECISION,

    lat DOUBLE PRECISION,

    heading INTEGER,

    course DOUBLE PRECISION,

    speed DOUBLE PRECISION

) PARTITION BY HASH (id);

/\* Δημιουργία υποπινάκων \*/

CREATE TABLE Positions\_Part1

PARTITION OF Positions\_Hash\_Partition

FOR VALUES WITH (MODULUS 4, REMAINDER 0);

CREATE TABLE Positions\_Part2

PARTITION OF Positions\_Hash\_Partition

FOR VALUES WITH (MODULUS 4, REMAINDER 1);

CREATE TABLE Positions\_Part3

PARTITION OF Positions\_Hash\_Partition

FOR VALUES WITH (MODULUS 4, REMAINDER 2);

CREATE TABLE Positions\_Part4

PARTITION OF Positions\_Hash\_Partition

FOR VALUES WITH (MODULUS 4, REMAINDER 3);

/\* Δημιουργία index \*/

CREATE INDEX pos\_part1\_t\_index ON Positions\_Part1 (t);

CREATE INDEX pos\_part1\_speed\_index ON Positions\_Part1 (speed);

CREATE INDEX pos\_part2\_t\_index ON Positions\_Part2 (t);

CREATE INDEX pos\_part2\_speed\_index ON Positions\_Part2 (speed);

CREATE INDEX pos\_part3\_t\_index ON Positions\_Part3 (t);

CREATE INDEX pos\_part3\_speed\_index ON Positions\_Part3 (speed);

CREATE INDEX pos\_part4\_t\_index ON Positions\_Part4 (t);

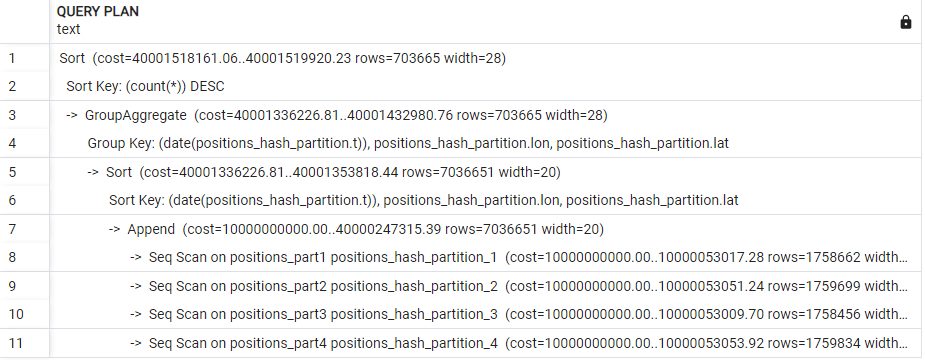
CREATE INDEX pos\_part4\_speed\_index ON Positions\_Part4 (speed);

/\* Γέμισμα πινάκων \*/

INSERT INTO Positions\_Hash\_Partition

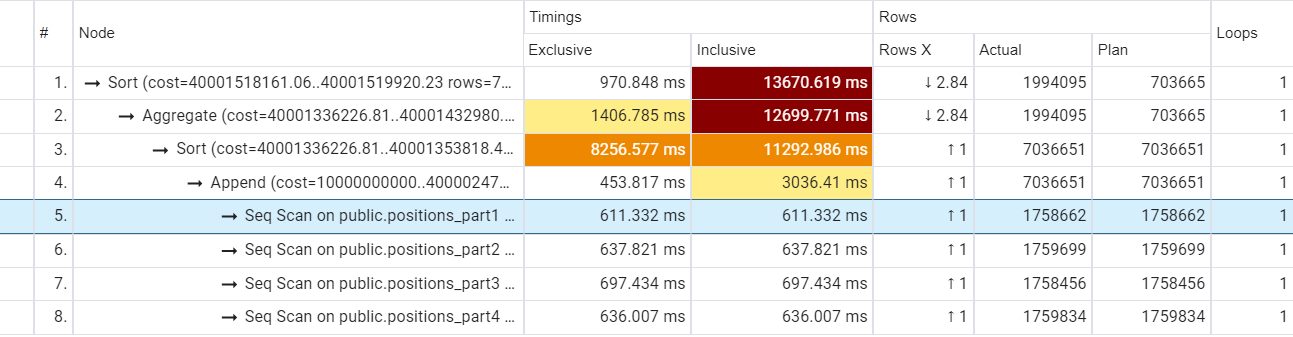
SELECT \* FROM Positions;

## Πλάνα εκτέλεσης

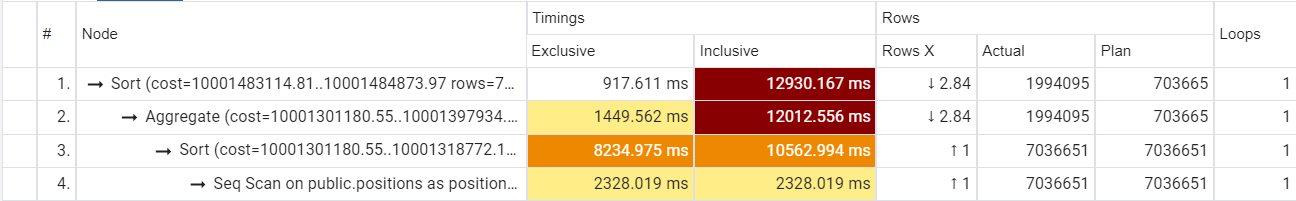


Εικόνα 0.1: Πλάνο εκτέλεσης query 1 για το ερώτημα 5

Εδώ παρατηρούμε ότι η PostgreSQL ΔΕΝ κάνει Parallel Sequential Scanάρα ενδέχεται και ο χρόνος εκτέλεσης να αυξηθεί από τα προηγούμενα ερωτήματα. Το βλέπουμε και παρακάτω με την εντολή EXPLAIN ANALYZE:



Εικόνα 0.2: Πλάνο + χρόνοι εκτέλεσης για το query 1 ME PARTITION

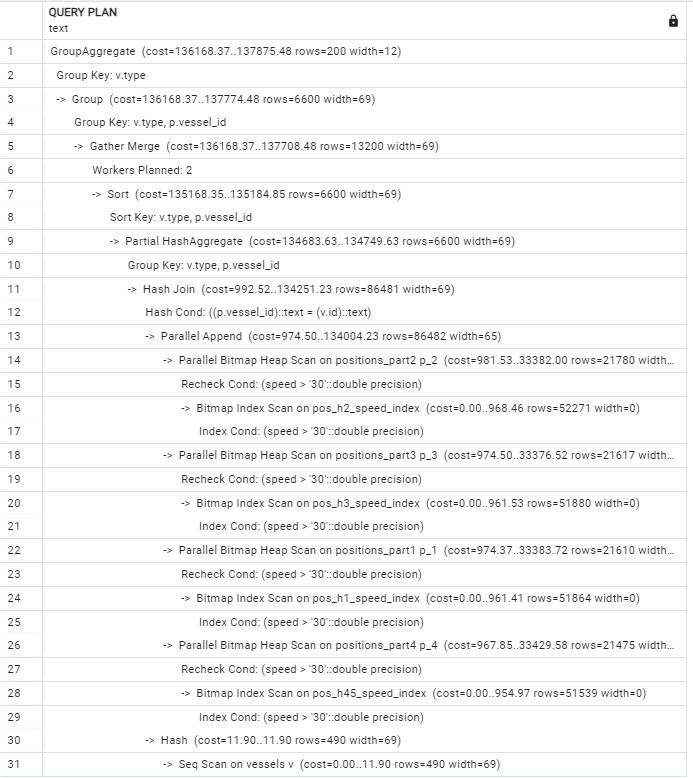


Εικόνα 0.3: Πλάνο + χρόνοι εκτέλεσης για το query 1 ΧΩΡΙΣ PARTITION

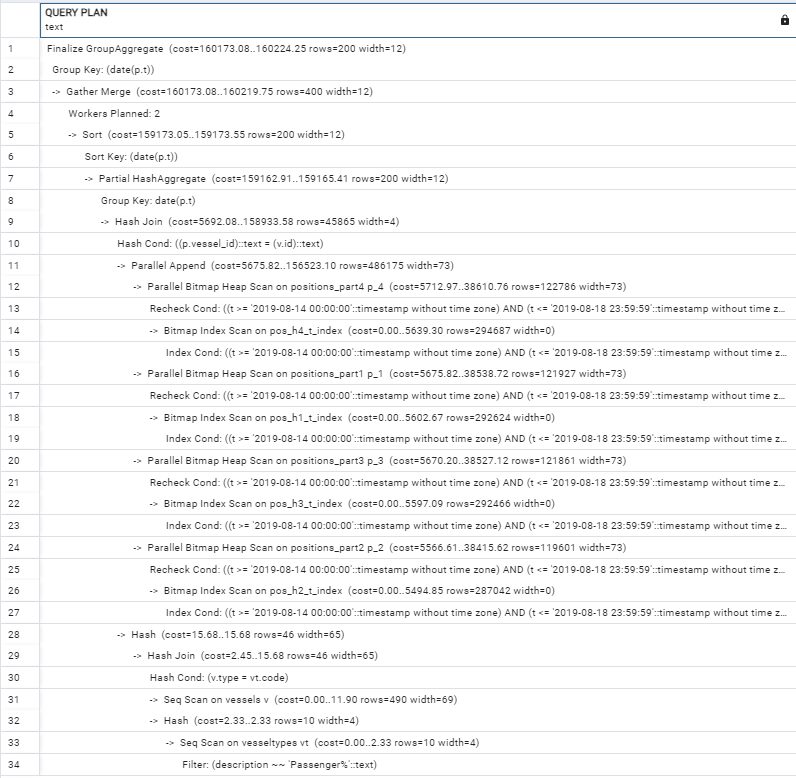
Από τα παραπάνω βλέπουμε ότι ο χρόνος για το Sequential Scan με διαμέριση είναι μεγαλύτερος από το χρόνο χωρίς!

**Για το query 2 δεν υπάρχει αλλαγή στο πλάνο εκτέλεσης, επειδή αφορά ΜΟΝΟ τον πίνακα Vessels**.

**ΣΗΜΕΙΩΣΗ:** Για τα παρακάτω πλάνα, μετά από επανεκκίνηση της PostgreSQL, τα max\_parallel\_workers\_per\_gather ξαναέγιναν 2. Όπου λοιπόν γράφει Workers Planned: 2 και όχι 5, είναι για αυτό το λόγο! (Στους χρόνους εκτέλεσης το φτιάξαμε)



Εικόνα 0.4: Πλάνο εκτέλεσης query 3 για το ερώτημα 5



Εικόνα 0.5: Πλάνο εκτέλεσης query 4 για το ερώτημα 5



Εικόνα 0.6: Πλάνο εκτέλεσης query 5A για το ερώτημα 5



Εικόνα 0.7: Πλάνο εκτέλεσης query 5B για το ερώτημα 5

## Χρόνοι Εκτέλεσης

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query** | **Χρόνος (sec)** |  |  | **Βελτίωση** |
|  | Mary | George | Average |  |
| 1 | #### | #### | #### | #### |
| 2 | #### | #### | #### | #### |
| 3 | 0,286 | 0,175 | 0,2305 | -16,94% |
| 4 | 0,312 | 0,266 | 0,289 | -30,02% |
| 5A | 0,258 | 0,167 | 0,2125 | -23,15% |
| 5B | 0,356 | 0,269 | 0,3125 | -27,75% |

Εκτός των πρώτων δύο ερωτημάτων, παρατηρούμε αρκετή βελτίωση στους χρόνους εκτέλεσης σε σχέση με το [ερώτημα 4](#_Χρόνοι_εκτέλεσης_1). Αυτό όπως είδαμε στα παραπάνω πλάνα εκτέλεσης συμβαίνει χάρη στα **Parallel Index Scan**.