ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ - ΠΛΗ303

Φάση Β' - Αναφορά

Ομάδα:

Μόδεστος Σιώτος	2016030030
Αλέξανδρος Γοριδάρης	2019030108

Όσον αφορά το 1ο μέρος της εκφώνησης, το έχουμε συμπεριλάβει στο .zip αρχείο και δεν μας ζητείται κάποια επεξήγηση για αυτό στην αναφορά μας.

Στο 2ο μέρος, μας ζητήθηκε να υλοποιήσουμε ένα ερώτημα σε PostgreSQL και να μελετήσουμε την απόδοσή του και πώς αυτή μπορεί να βελτιωθεί με τη βοήθεια της εντολής EXPLAIN ANALYZE, τη χρήση ευρετηρίων, clusters, την απενεργοποίηση αλγορίθμων υπολογισμού συνδέσεων και την αλλαγή σειράς των συνδέσεων.

Αρχικά, τρέχουμε το ερώτημα χωρίς κάποιο ευρετήριο ή κάποια άλλη αλλαγή, μόνο με τη χρήση του *explain analyze*.

```
SET max_parallel_workers_per_gather = 0;

EXPLAIN ANALYZE
SELECT e."eduLevel", COUNT(*)
FROM education e
JOIN (
    SELECT a.email
    FROM advertisement a
    JOIN "jobOffer" j ON a."advertisementID" = j."advertisementID"
    WHERE a."datePosted" >= CURRENT_DATE - INTERVAL '6 months'
    AND j."fromAge" > 21 AND j."toAge" < 30
    GROUP BY a.email
    HAVING COUNT(a."advertisementID") >= 2
) AS valid_ads ON e.email = valid_ads.email
JOIN (
    SELECT m."receiverEmail"
    FROM msg m
    WHERE m."dateSent" >= CURRENT_DATE - INTERVAL '6 months'
    GROUP BY m."receiverEmail"
) AS recent_msgs ON e.email = recent_msgs."receiverEmail"
WHERE e.country = 'Canada'
GROUP BY e."eduLevel";
```

Το Data Output είναι το παρακάτω:

	QUERY PLAN text
1	GroupAggregate (cost=105606.47..105608.75 rows=4 width=23) (actual time=1039.343..1039.413 rows=4 loops=1)
2	Group Key: e."eduLevel"
3	-> Sort (cost=105606.47..105607.22 rows=299 width=15) (actual time=1039.313..1039.343 rows=225 loops=1)
4	Sort Key: e."eduLevel"
5	Sort Method: quicksort Memory: 38kB
6	-> Hash Join (cost=51772.44..105594.18 rows=299 width=15) (actual time=587.499..1039.057 rows=225 loops=1)
7	Hash Cond: ((e.email)::text = (m."receiverEmail")::text)
8	-> Hash Join (cost=22353.10..76163.38 rows=4367 width=55) (actual time=221.204..667.113 rows=257 loops=1)
9	Hash Cond: ((e.email)::text = (a.email)::text)
10	-> Seq Scan on education e (cost=0.00..50327.26 rows=1326853 width=35) (actual time=0.475..315.012 rows=1330542 loops=1)
11	Filter: ((country)::text = 'Canada'::text)
12	Rows Removed by Filter: 553479
13	-> Hash (cost=22321.87..22321.87 rows=2499 width=20) (actual time=218.604..218.607 rows=164 loops=1)
14	Buckets: 4096 Batches: 1 Memory Usage: 41kB
15	-> HashAggregate (cost=22203.18..22296.88 rows=2499 width=20) (actual time=218.013..218.560 rows=164 loops=1)
16	Group Key: a.email
17	Filter: (count(a."advertisementID") >= 2)
18	Batches: 1 Memory Usage: 1169kB
19	Rows Removed by Filter: 5418
20	-> Hash Join (cost=6378.68..22165.70 rows=7496 width=24) (actual time=19.834..214.139 rows=5746 loops=1)
21	Hash Cond: (a."advertisementID" = j."advertisementID")
22	-> Seq Scan on advertisement a (cost=0.00..15276.77 rows=194380 width=24) (actual time=0.025..163.256 rows=193066 loop…
23	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
24	Rows Removed by Filter: 229321
25	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=19.494..19.495 rows=12431 loops=1)
26	Buckets: 16384 Batches: 1 Memory Usage: 566kB
27	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.031..17.699 rows=12431 loops=1)
28	Filter: (("fromAge" > 21) AND ("toAge" < 30))
29	Rows Removed by Filter: 198574
30	-> Hash (cost=28769.80..28769.80 rows=51963 width=20) (actual time=360.509..360.510 rows=66577 loops=1)
31	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
32	-> HashAggregate (cost=27730.54..28250.17 rows=51963 width=20) (actual time=301.967..345.516 rows=66577 loops=1)
33	Group Key: m."receiverEmail"
34	Batches: 5 Memory Usage: 4145kB Disk Usage: 7664kB
35	-> Seq Scan on msg m (cost=0.00..26938.43 rows=316846 width=20) (actual time=0.033..204.959 rows=317198 loops=1)
36	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
37	Rows Removed by Filter: 377512
38	Planning Time: 0.377 ms
39	Execution Time: 1043.876 ms
Total rows: 39 of 39	Query complete 00:00:01.110
38	Planning Time: 0.669 ms
39	Execution Time: 1081.444 ms
Total rows: 39 of 39	Query complete 00:00:01.105
38	Planning Time: 0.546 ms
39	Execution Time: 1091.521 ms
Total rows: 39 of 39	Query complete 00:00:01.115

Από την ανάλυση λαμβάνουμε τις εξής πληροφορίες:

Στις πρώτες σειρές γίνεται το τελικό sorting και aggregation με βάση το κλειδί *e.eduLevel*. Το Actual Time του sort είναι 1039.313 ms και προσπελαύνει 225 rows. Έπειτα έχουμε το Hash Join με το subquery recent_msgs με Hash Condition το email και το Hash Join με το subquery valid_ads πάλι με βάση το email. Μετά έχουμε το sequential scan στο education για το country = 'Canada'. Ενδεικτικά με αυτό το φίλτρο διαγράφηκαν 553.479 rows. Άλλες παρατηρήσεις άξιες σχολιασμού είναι το sequential scan που γίνεται στο advertisement a, για να φιλτραριστούν οι αγγελίες που έχουν δημοσιευτεί τους τελευταίους 6 μήνες, το sequential scan στο jobOffer j για να έχουμε το ζητούμενο age group και τέλος το sequential scan στο msg m για να φιλτράρουμε ότι έχει δεχτεί μηνύματα τους τελευταίους 6 μήνες.

Ευρετήρια

Η 1η μας σκέψη αφορά, το τί είδους ευρετήρια θα χρησιμοποιήσουμε για να επιταχύνουμε την εκτέλεση του αιτήματος.
Γενικά, τα B-Tree indexes είναι περισσότερο αποδοτικά καθώς προσφέρουν μεγαλύτερη ευελιξία και μπορούν να χρησιμοποιηθούν για συγκρίσεις και εύρος τιμών. Αντίθετα, τα Hash indexes χρησιμοποιούνται κυρίως σε σχέσεις ισότητας.

Όσον αφορά τον πίνακα Education, θα χρειαστούμε ένα ευρετήριο στο email αφού αυτό χρησιμοποιείται για τη σύνδεση με τον πίνακα advertisement και msg και για να φιλτραριστούν τα αποτελέσματα στην υποερώτηση WHERE e.country = 'Canada'. Δοκιμάσαμε και Hash και B-Tree ευρετήρια και αποδείχθηκε ότι ο Query Planner, προτιμά τον Hash στην προκειμένη περίπτωση. Αυτό είναι λογικό, γιατί έχουμε σχέσεις ισότητας και ακριβείς αναζητήσεις. Έτσι κάνουμε DROP το B-Tree ευρετήριο και κρατάμε το Hash.

```
CREATE INDEX education_email_idx ON education USING hash (email);
```

24	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.003..0.003 rows=2 loops...
25	Index Cond: ((email)::text = (a.email)::text)
26	Filter: ((country)::text = 'Canada'::text)
27	Rows Removed by Filter: 1
28	-> Hash (cost=28769.80..28769.80 rows=51963 width=20) (actual time=512.722..512.723 rows=66577 loops=1)
29	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
30	-> HashAggregate (cost=27730.54..28250.17 rows=51963 width=20) (actual time=443.466..497.481 rows=66577 loops=1)
31	Group Key: m."receiverEmail"
32	Batches: 5 Memory Usage: 4145kB Disk Usage: 7664kB
33	-> Seq Scan on msg m (cost=0.00..26938.43 rows=316846 width=20) (actual time=0.075..321.915 rows=317198 loops=1)
34	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
35	Rows Removed by Filter: 377512
36	Planning Time: 0.447 ms
37	Execution Time: 809.558 ms

Total rows: 37 of 37 Query complete 00:00:00.860

Πράγματι, βλέπουμε στο output το index scan και έχουμε:
Planning Time: 0.447 ms
Execution Time: 809.558 ms

Έπειτα, θα δημιουργήσουμε δενδρικό ευρετήριο για τον πίνακα advertisement στο datePosted. Το ερώτημα περιλαμβάνει τη συνθήκη a."datePosted" >= CURRENT_DATE - INTERVAL '6 months'. Αυτή η συνθήκη φιλτράρει τις αγγελίες που έχουν αναρτηθεί τους τελευταίους 6 μήνες.
Χωρίς ευρετήριο, το PostgreSQL πρέπει να διατρέξει ολόκληρο τον πίνακα advertisement (sequential scan) για να βρει τις γραμμές που πληρούν τη συνθήκη αυτή, κάτι που είναι χρονοβόρο για μεγάλο πίνακα.
Με το ευρετήριο στο πεδίο datePosted, το PostgreSQL μπορεί να χρησιμοποιήσει το ευρετήριο για να βρει γρήγορα τις γραμμές που πληρούν τη συνθήκη χρονικού διαστήματος. Το δενδρικό ευρετήριο (B-Tree) σε αντίθεση με το ευρετήριο κατακερματισμού (hash), είναι εξαιρετικά αποδοτικό για εύρος αναζητήσεων όπως αυτή.

```
CREATE INDEX advertisement_date_idx ON advertisement("datePosted");
```

	QUERY PLAN text
13	Rows Removed by Filter: 5418
14	-> Hash Join (cost=8569.55..20366.45 rows=7496 width=24) (actual time=37.091..101.045 rows=5746 loops=1)
15	Hash Cond: (a."advertisementID" = j."advertisementID")
16	-> Bitmap Heap Scan on advertisement a (cost=2190.87..13477.52 rows=194380 width=24) (actual time=8.482..40.881 rows=193066 loops=1)
17	Recheck Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
18	Heap Blocks: exact=7885
19	-> Bitmap Index Scan on advertisement_date_idx (cost=0.00..2142.28 rows=194380 width=0) (actual time=7.553..7.553 rows=193066 loo...
20	Index Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
21	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=28.514..28.515 rows=12431 loops=1)
22	Buckets: 16384 Batches: 1 Memory Usage: 566kB
23	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.039..26.015 rows=12431 loops=1)
24	Filter: (("fromAge" > 21) AND ("toAge" < 30))
25	Rows Removed by Filter: 198574
26	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.004..0.004 rows=2 loops=164)
27	Index Cond: ((email)::text = (a.email)::text)
28	Filter: ((country)::text = 'Canada'::text)
29	Rows Removed by Filter: 1
30	-> Hash (cost=28769.80..28769.80 rows=51963 width=20) (actual time=571.945..571.946 rows=66577 loops=1)
31	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
32	-> HashAggregate (cost=27730.54..28250.17 rows=51963 width=20) (actual time=501.880..553.277 rows=66577 loops=1)
33	Group Key: m."receiverEmail"
34	Batches: 5 Memory Usage: 4145kB Disk Usage: 7664kB
35	-> Seq Scan on msg m (cost=0.00..26938.43 rows=316846 width=20) (actual time=1.007..364.300 rows=317198 loops=1)
36	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
37	Rows Removed by Filter: 377512
38	Planning Time: 0.653 ms
39	Execution Time: 691.997 ms

Total rows: 39 of 39 Query complete 00:00:00.723

Planning Time: 0.653 ms
Execution Time: 691.997 ms

Μετά, κάνουμε DROP το προηγούμενο ευρετήριο για να τρέξουμε χωριστά ένα καινούργιο ευρετήριο για τον πίνακα msg στο dateSent. Και εδώ το δενδρικό ευρετήριο είναι το πιο κατάλληλο, για τους λόγους που αναφέρθηκαν παραπάνω.

```
CREATE INDEX msg_date_idx ON msg("dateSent");
```

33	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=35.861..158.333 rows=317198 loops=...
34	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
35	Heap Blocks: exact=14781
36	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=32.385..32.385 rows=317198 loo...
37	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
38	Planning Time: 0.358 ms
39	Execution Time: 622.122 ms

Total rows: 39 of 39 | Query complete 00:00:00.674

Planning Time: 0.358 ms
Execution Time: 622.122 ms

Τέλος, θα χρησιμοποιήσουμε και τα 3 ευρετήρια μαζί:

```
CREATE INDEX education_email_idx ON education USING hash (email);
CREATE INDEX advertisement_date_idx ON advertisement("datePosted");
CREATE INDEX msg_date_idx ON msg("dateSent");
```

	QUERY PLAN text
15	Hash Cond: (a."advertisementID" = j."advertisementID")
16	-> Bitmap Heap Scan on advertisement a (cost=2190.87..13477.52 rows=194380 width=24) (actual time=11.092..89.732 rows=193066 loops=1)
17	Recheck Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
18	Heap Blocks: exact=7885
19	-> Bitmap Index Scan on advertisement_date_idx (cost=0.00..2142.28 rows=194380 width=0) (actual time=10.169..10.169 rows=193066 loo...
20	Index Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
21	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=56.340..56.341 rows=12431 loops=1)
22	Buckets: 16384 Batches: 1 Memory Usage: 566kB
23	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.057..51.767 rows=12431 loops=1)
24	Filter: (("fromAge" > 21) AND ("toAge" < 30))
25	Rows Removed by Filter: 198574
26	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.029..0.034 rows=2 loops=164)
27	Index Cond: ((email)::text = (a.email)::text)
28	Filter: ((country)::text = 'Canada'::text)
29	Rows Removed by Filter: 1
30	-> Hash (cost=25729.17..25729.17 rows=51963 width=20) (actual time=358.820..358.821 rows=66577 loops=1)
31	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
32	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=288.841..341.003 rows=66577 loops=1)
33	Group Key: m."receiverEmail"
34	Batches: 5 Memory Usage: 4145kB Disk Usage: 7664kB
35	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=18.748..171.353 rows=317198 loops=1)
36	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
37	Heap Blocks: exact=14781
38	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=17.000..17.001 rows=317198 loops=1)
39	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
40	Planning Time: 0.864 ms
41	Execution Time: 557.844 ms

Total rows: 41 of 41 | Query complete 00:00:00.587

Φαίνεται να προκύπτουν τα καλύτερα αποτελέσματα:
Planning Time: 0.864 ms
Execution Time: 557.844 ms

Clustering

Αφού καταλήξαμε σε ευρετήρια, θα ελέγξουμε αν η δυνατότητα ομαδοποίησης (clustering) θα μας βοηθήσει να επιταχύνουμε κι περαιτέρω το ερώτημα.
Η PostgreSQL δεν υποστηρίζει clustering στα Hash Indexes.
Θα δοκιμάσουμε για τα 2 δενδρικά ευρετήρια.

Μόνο για το advertisement:

```
CLUSTER advertisement USING advertisement_date_idx;
```

	QUERY PLAN text
15	Hash Cond: (a."advertisementID" = j."advertisementID")
16	-> Bitmap Heap Scan on advertisement a (cost=2190.87..13483.52 rows=194380 width=24) (actual time=6.902..48.718 rows=193066 loops=1)
17	Recheck Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
18	Heap Blocks: exact=3608
19	-> Bitmap Index Scan on advertisement_date_idx (cost=0.00..2142.28 rows=194380 width=0) (actual time=6.551..6.551 rows=193066 loo…
20	Index Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
21	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=44.386..44.386 rows=12431 loops=1)
22	Buckets: 16384 Batches: 1 Memory Usage: 566kB
23	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.055..41.393 rows=12431 loops=1)
24	Filter: (("fromAge" > 21) AND ("toAge" < 30))
25	Rows Removed by Filter: 198574
26	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.025..0.029 rows=2 loops=164)
27	Index Cond: ((email)::text = (a.email)::text)
28	Filter: ((country)::text = 'Canada'::text)
29	Rows Removed by Filter: 1
30	-> Hash (cost=25729.17..25729.17 rows=51963 width=20) (actual time=401.401..401.402 rows=66577 loops=1)
31	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
32	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=332.559..384.704 rows=66577 loops=1)
33	Group Key: m."receiverEmail"
34	Batches: 5 Memory Usage: 4145kB Disk Usage: 7664kB
35	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=20.925..196.790 rows=317198 loops=1)
36	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
37	Heap Blocks: exact=14781
38	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=18.791..18.791 rows=317198 loops=1)
39	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
40	Planning Time: 0.683 ms
41	Execution Time: 542.327 ms

Total rows: 41 of 41 | Query complete 00:00:00.590

Planning Time: 0.683 ms
Execution Time: 542.327 ms

Μόνο για το msg:

```
CLUSTER msg USING msg_date_idx;
```

	QUERY PLAN text
15	Hash Cond: (a."advertisementID" = j."advertisementID")
16	-> Bitmap Heap Scan on advertisement a (cost=2190.87..13483.52 rows=194380 width=24) (actual time=9.664..33.414 rows=193066 loops=1)
17	Recheck Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
18	Heap Blocks: exact=3608
19	-> Bitmap Index Scan on advertisement_date_idx (cost=0.00..2142.28 rows=194380 width=0) (actual time=9.255..9.255 rows=193066 loo...
20	Index Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
21	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=37.279..37.281 rows=12431 loops=1)
22	Buckets: 16384 Batches: 1 Memory Usage: 566kB
23	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.147..34.150 rows=12431 loops=1)
24	Filter: (("fromAge" > 21) AND ("toAge" < 30))
25	Rows Removed by Filter: 198574
26	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.002..0.003 rows=2 loops=164)
27	Index Cond: ((email)::text = (a.email)::text)
28	Filter: ((country)::text = 'Canada'::text)
29	Rows Removed by Filter: 1
30	-> Hash (cost=25729.17..25729.17 rows=51963 width=20) (actual time=236.293..236.295 rows=66577 loops=1)
31	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
32	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=177.244..220.892 rows=66577 loops=1)
33	Group Key: m."receiverEmail"
34	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
35	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=9.230..44.908 rows=317198 loops=1)
36	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
37	Heap Blocks: exact=6750
38	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=7.842..7.842 rows=317198 loops=1)
39	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
40	Planning Time: 0.846 ms
41	Execution Time: 354.638 ms
Total rows: 41 of 41	Query complete 00:00:00.404

Planning Time: 0.846 ms
Execution Time: 354.638 ms

Και για advertisement και για msg:

```
CLUSTER advertisement USING advertisement_date_idx;
CLUSTER msg USING msg_date_idx;
```

40	Planning Time: 0.497 ms
41	Execution Time: 303.631 ms
Total rows: 41 of 41	Query complete 00:00:00.342
40	Planning Time: 0.582 ms
41	Execution Time: 293.387 ms
Total rows: 41 of 41	Query complete 00:00:00.344
40	Planning Time: 0.357 ms
41	Execution Time: 310.361 ms
Total rows: 41 of 41	Query complete 00:00:00.350

Συμπεραίνουμε ότι χρησιμοποιώντας ομαδοποίηση (clustering) και για το advertisement και για το msg επιταχύνεται η ερώτηση στο μέγιστο.

Απενεργοποίηση αλγορίθμων υπολογισμού συνδέσεων

Οι αλγόριθμοι που μπορούμε να απενεργοποιήσουμε περιλαμβάνουν:

- Nested Loop Joins

- Hash Joins
- Merge Joins

Παρατηρούμε ότι δεν έχουμε κάποιο Merge Join, οπότε θα απενεργοποιήσουμε τους άλλους 2 αρχικά έναν έναν και μετά και τους 2 μαζί.

Απενεργοποίηση Nested Loop Join (Διατηρώντας τα ευρετήρια και clusters)

```
SET enable_nestloop = off;
```

	QUERY PLAN text
17	Filter: (count(a."advertisementID") >= 2)
18	Batches: 1 Memory Usage: 1169kB
19	Rows Removed by Filter: 5418
20	-> Hash Join (cost=8569.55..20372.45 rows=7496 width=24) (actual time=23.004..58.154 rows=5746 loops=1)
21	Hash Cond: (a."advertisementID" = j."advertisementID")
22	-> Bitmap Heap Scan on advertisement a (cost=2190.87..13483.52 rows=194380 width=24) (actual time=4.451..20.047 rows=193066 loops=1)
23	Recheck Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
24	Heap Blocks: exact=3608
25	-> Bitmap Index Scan on advertisement_date_idx (cost=0.00..2142.28 rows=194380 width=0) (actual time=4.066..4.066 rows=193066 loo...
26	Index Cond: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
27	-> Hash (cost=6175.07..6175.07 rows=16288 width=4) (actual time=18.522..18.523 rows=12431 loops=1)
28	Buckets: 16384 Batches: 1 Memory Usage: 566kB
29	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.047..16.912 rows=12431 loops=1)
30	Filter: (("fromAge" > 21) AND ("toAge" < 30))
31	Rows Removed by Filter: 198574
32	-> Hash (cost=25729.17..25729.17 rows=51963 width=20) (actual time=231.243..231.244 rows=66577 loops=1)
33	Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3329kB
34	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=164.387..214.261 rows=66577 loops=1)
35	Group Key: m."receiverEmail"
36	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
37	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=6.951..40.567 rows=317198 loops=1)
38	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
39	Heap Blocks: exact=6750
40	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=6.295..6.295 rows=317198 loops=1)
41	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
42	Planning Time: 0.370 ms
43	Execution Time: 818.890 ms

Total rows: 43 of 43 Query complete 00:00:00.858

Παρατηρούμε μείωση του Planning Time άλλα αύξηση του Execution Time.
Planning Time: 0.370 ms
Execution Time: 818.890 ms

Απενεργοποίηση Hash Join (Διατηρώντας τα ευρετήρια και clusters)

```
SET enable_hashjoin = off;
```

	QUERY PLAN text
17	Merge Cond: (a."advertisementID" = j."advertisementID")
18	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..22049.74 rows=194380 width=24) (actual time=0.008..257.731 rows=193048 loo…
19	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
20	Rows Removed by Filter: 229304
21	-> Sort (cost=7314.54..7355.26 rows=16288 width=4) (actual time=33.600..34.490 rows=12431 loops=1)
22	Sort Key: j."advertisementID"
23	Sort Method: quicksort Memory: 967kB
24	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.053..32.443 rows=12431 loops=1)
25	Filter: (("fromAge" > 21) AND ("toAge" < 30))
26	Rows Removed by Filter: 198574
27	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.023..0.028 rows=2 loops=164)
28	Index Cond: ((email)::text = (a.email)::text)
29	Filter: ((country)::text = 'Canada'::text)
30	Rows Removed by Filter: 1
31	-> Sort (cost=29799.22..29929.13 rows=51963 width=20) (actual time=531.923..551.203 rows=65124 loops=1)
32	Sort Key: m."receiverEmail"
33	Sort Method: external merge Disk: 1960kB
34	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=222.715..264.168 rows=66577 loops=1)
35	Group Key: m."receiverEmail"
36	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
37	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=9.315..88.155 rows=317198 loops=1)
38	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
39	Heap Blocks: exact=6750
40	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=8.656..8.656 rows=317198 loops=1)
41	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
42	Planning Time: 0.423 ms
43	Execution Time: 898.849 ms
Total rows: 43 of 43	Query complete 00:00:00.948

Παρατηρούμε ότι δεν εμφανίζεται ιδιαίτερη μείωση στο Planning Time, ενώ κι εδώ το Execution Time είναι σημαντικά μεγαλύτερο.
Planning Time: 0.423 ms
Execution Time: 898.849 ms

Απενεργοποίηση Nested Loop Join και Hash Join (Διατηρώντας τα ευρετήρια και clusters)

```
SET enable_nestloop = off;
SET enable_hashjoin = off;
```

	QUERY PLAN text
22	Batches: 1 Memory Usage: 1169kB
23	Rows Removed by Filter: 5418
24	-> Merge Join (cost=7315.66..30006.64 rows=7496 width=24) (actual time=38.175..313.279 rows=5746 loops=1)
25	Merge Cond: (a."advertisementID" = j."advertisementID")
26	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..22049.74 rows=194380 width=24) (actual time=0.028..263.044 rows=193048 loo…
27	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
28	Rows Removed by Filter: 229304
29	-> Sort (cost=7314.54..7355.26 rows=16288 width=4) (actual time=37.726..38.732 rows=12431 loops=1)
30	Sort Key: j."advertisementID"
31	Sort Method: quicksort Memory: 967kB
32	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.108..35.892 rows=12431 loops=1)
33	Filter: (("fromAge" > 21) AND ("toAge" < 30))
34	Rows Removed by Filter: 198574
35	-> Sort (cost=29799.22..29929.13 rows=51963 width=20) (actual time=497.610..517.409 rows=65124 loops=1)
36	Sort Key: m."receiverEmail"
37	Sort Method: external merge Disk: 1960kB
38	-> HashAggregate (cost=24689.91..25209.54 rows=51963 width=20) (actual time=197.012..238.958 rows=66577 loops=1)
39	Group Key: m."receiverEmail"
40	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
41	-> Bitmap Heap Scan on msg m (cost=3571.99..23897.79 rows=316846 width=20) (actual time=10.098..78.997 rows=317198 loops=1)
42	Recheck Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
43	Heap Blocks: exact=6750
44	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=9.425..9.425 rows=317198 loops=1)
45	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
46	Planning Time: 0.701 ms
47	Execution Time: 8728.718 ms

Total rows: 47 of 47 | Query complete 00:00:08.782

Έχοντας απενεργοποιήσει και τους 2 αλγόριθμους υπολογισμού συνδέσεων παρατηρούμε ότι το Planing Time αυξάνεται ελαφρώς και το Execution Time εμφανίζει τεράστια αύξηση.
Planning Time: 0.701 ms
Execution Time: 8728.718 ms

Βλέπουμε ότι όταν απενεργοποιούμε τους 2 αλγόριθμους, χρησιμοποιείται ο Merge Join. Θα δοκιμάσουμε να τον απενεργοποιήσουμε και αυτόν.

Απενεργοποίηση Nested Loop Join, Hash Join και Merge Join(Διατηρώντας τα ευρετήρια και clusters)

```
SET enable_nestloop = off;
SET enable_hashjoin = off;
SET enable_mergejoin = off;
```

	QUERY PLAN text
15	-> Bitmap Index Scan on msg_date_idx (cost=0.00..3492.78 rows=316846 width=0) (actual time=6.695..6.696 rows=317198 loops=1)
16	Index Cond: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
17	-> Materialize (cost=20000050158.39..20000077449.08 rows=4367 width=55) (actual time=0.001..0.009 rows=257 loops=66577)
18	-> Nested Loop (cost=20000050158.39..20000077427.24 rows=4367 width=55) (actual time=82.498..87.971 rows=257 loops=1)
19	-> GroupAggregate (cost=10000050158.39..10000050308.31 rows=2499 width=20) (actual time=82.442..84.245 rows=164 loops=1)
20	Group Key: a.email
21	Filter: (count(a."advertisementID") >= 2)
22	Rows Removed by Filter: 5418
23	-> Sort (cost=10000050158.39..10000050177.13 rows=7496 width=24) (actual time=82.399..82.854 rows=5746 loops=1)
24	Sort Key: a.email
25	Sort Method: quicksort Memory: 641kB
26	-> Nested Loop (cost=10000000000.42..10000049675.96 rows=7496 width=24) (actual time=0.067..65.314 rows=5746 loops=1)
27	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.035..31.468 rows=12431 loops=1)
28	Filter: (("fromAge" > 21) AND ("toAge" < 30))
29	Rows Removed by Filter: 198574
30	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..2.67 rows=1 width=24) (actual time=0.002..0.002 rows=0 loops=1...
31	Index Cond: ("advertisementID" = j."advertisementID")
32	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
33	Rows Removed by Filter: 1
34	-> Index Scan using education_email_idx on education e (cost=0.00..10.82 rows=2 width=35) (actual time=0.019..0.022 rows=2 loops=164)
35	Index Cond: ((email)::text = (a.email)::text)
36	Filter: ((country)::text = 'Canada'::text)
37	Rows Removed by Filter: 1
38	Planning Time: 0.386 ms
39	Execution Time: 1869.227 ms

Total rows: 39 of 39 Query complete 00:00:01.912

Παρατηρούμε ότι ενώ έχουμε απενεργοποιήσει και τους 3 αλγόριθμους σύνδεσης, τελικά χρησιμοποιείται ο αλγόριθμος Nested Loop Join. Το Planning time είναι παρόμοιο με το προηγούμενο και το Execution Time είναι εμφανώς αυξημένο, αλλά λιγότερο από την ακριβώς προηγούμενη περίπτωση.
Planning Time: 0.386 ms
Execution Time: 1869.227 ms

Συμπεραίνουμε ότι δεν πρέπει να απενεργοποιήσουμε κανέναν από τους αλγορίθμους σύνδεσης, καθώς είναι κρίσιμοι για την αποδοτική εκτέλεση του ερωτήματος μας.

Στο αρχικό μας ερώτημα, χωρίς τα ευρετήρια και τα clusters, έχουμε μόνο Hash Joins.

Απενεργοποίηση Hash Join (Χωρίς τα ευρετήρια και clusters)

	QUERY PLAN text
19	-> HashAggregate (cost=30044.12..30137.82 rows=2499 width=20) (actual time=257.762..258.308 rows=164 loops=1)
20	Group Key: a.email
21	Filter: (count(a."advertisementID") >= 2)
22	Batches: 1 Memory Usage: 1169kB
23	Rows Removed by Filter: 5418
24	-> Merge Join (cost=7315.66..30006.64 rows=7496 width=24) (actual time=18.787..254.889 rows=5746 loops=1)
25	Merge Cond: (a."advertisementID" = j."advertisementID")
26	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..22049.74 rows=194380 width=24) (actual time=0.021..225.336 rows=193048 loo...
27	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
28	Rows Removed by Filter: 229304
29	-> Sort (cost=7314.54..7355.26 rows=16288 width=4) (actual time=18.396..19.233 rows=12431 loops=1)
30	Sort Key: j."advertisementID"
31	Sort Method: quicksort Memory: 967kB
32	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.036..16.813 rows=12431 loops=1)
33	Filter: (("fromAge" > 21) AND ("toAge" < 30))
34	Rows Removed by Filter: 198574
35	-> Sort (cost=32839.85..32969.76 rows=51963 width=20) (actual time=604.445..624.829 rows=65124 loops=1)
36	Sort Key: m."receiverEmail"
37	Sort Method: external merge Disk: 1960kB
38	-> HashAggregate (cost=27730.54..28250.17 rows=51963 width=20) (actual time=328.208..362.606 rows=66577 loops=1)
39	Group Key: m."receiverEmail"
40	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
41	-> Seq Scan on msg m (cost=0.00..26938.43 rows=316846 width=20) (actual time=120.139..219.969 rows=317198 loops=1)
42	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
43	Rows Removed by Filter: 377512
44	Planning Time: 0.309 ms
45	Execution Time: 8402.380 ms

Total rows: 45 of 45 Query complete 00:00:08.435

Το Planning Time δεν παρουσιάζεται αυξημένο, σε αντίθεση με το Execution TIme που εμφανίζει πολύ μεγάλη αύξηση.
Planning Time: 0.309 ms
Execution Time: 8402.380 ms

Παρατηρούμε ότι όταν απενεργοποιούμε τα Hash Joins, χρησιμοποιούνται Merge Joins.

Απενεργοποίηση Hash Join και Merge Join (Χωρίς τα ευρετήρια και clusters)

	QUERY PLAN text
11	Batches: 5 Memory Usage: 4145kB Disk Usage: 7208kB
12	-> Seq Scan on msg m (cost=0.00..26938.43 rows=316846 width=20) (actual time=133.779..230.938 rows=317198 loops=1)
13	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
14	Rows Removed by Filter: 377512
15	-> Materialize (cost=49713.44..49837197.26 rows=4367 width=55) (actual time=0.002..0.307 rows=257 loops=66577)
16	-> Nested Loop (cost=49713.44..49837175.43 rows=4367 width=55) (actual time=129.273..19933.683 rows=257 loops=1)
17	Join Filter: ((e.email)::text = (a.email)::text)
18	Rows Removed by Join Filter: 218186431
19	-> Seq Scan on education e (cost=0.00..50327.26 rows=1326853 width=35) (actual time=0.036..452.343 rows=1330542 loops=1)
20	Filter: ((country)::text = 'Canada'::text)
21	Rows Removed by Filter: 553479
22	-> Materialize (cost=49713.44..49844.62 rows=2499 width=20) (actual time=0.000..0.005 rows=164 loops=1330542)
23	-> HashAggregate (cost=49713.44..49807.14 rows=2499 width=20) (actual time=43.138..43.695 rows=164 loops=1)
24	Group Key: a.email
25	Filter: (count(a."advertisementID") >= 2)
26	Batches: 1 Memory Usage: 1169kB
27	Rows Removed by Filter: 5418
28	-> Nested Loop (cost=0.42..49675.96 rows=7496 width=24) (actual time=0.047..41.044 rows=5746 loops=1)
29	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.030..16.446 rows=12431 loops=1)
30	Filter: (("fromAge" > 21) AND ("toAge" < 30))
31	Rows Removed by Filter: 198574
32	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..2.67 rows=1 width=24) (actual time=0.002..0.002 rows=0 loops=1...
33	Index Cond: ("advertisementID" = j."advertisementID")
34	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
35	Rows Removed by Filter: 1
36	Planning Time: 0.432 ms
37	Execution Time: 21819.102 ms

Total rows: 37 of 37 Query complete 00:00:22.344

Παρατηρούμε παρόμοιο Planning Time, αλλά ακόμα μεγαλύτερη αύξηση στο χρόνο εκτέλεσης.
Planning Time: 0.432 ms
Execution Time: 21819.102 ms

Παρατηρούμε ότι όταν απενεργοποιούμε τα Hash Joins και τα Merge Joins, χρησιμοποιούνται Nested Loop Joins.

Απενεργοποίηση Hash Join, Merge Join και Nested Loop Join (Χωρίς τα ευρετήρια και clusters)

	QUERY PLAN text
13	Filter: ("dateSent" >= (CURRENT_DATE - '6 mons'::interval))
14	Rows Removed by Filter: 377512
15	-> Materialize (cost=20000050158.39..20075748217.24 rows=4367 width=55) (actual time=0.007..0.515 rows=257 loops=66577)
16	-> Nested Loop (cost=20000050158.39..20075748195.40 rows=4367 width=55) (actual time=485.554..33797.014 rows=257 loops=1)
17	Join Filter: ((e.email)::text = (a.email)::text)
18	Rows Removed by Join Filter: 218208631
19	-> GroupAggregate (cost=10000050158.39..10000050308.31 rows=2499 width=20) (actual time=59.121..62.227 rows=164 loops=1)
20	Group Key: a.email
21	Filter: (count(a."advertisementID") >= 2)
22	Rows Removed by Filter: 5418
23	-> Sort (cost=10000050158.39..10000050177.13 rows=7496 width=24) (actual time=59.080..59.906 rows=5746 loops=1)
24	Sort Key: a.email
25	Sort Method: quicksort Memory: 641kB
26	-> Nested Loop (cost=10000000000.42..10000049675.96 rows=7496 width=24) (actual time=0.052..43.116 rows=5746 loops=1)
27	-> Seq Scan on "jobOffer" j (cost=0.00..6175.07 rows=16288 width=4) (actual time=0.035..16.918 rows=12431 loops=1)
28	Filter: (("fromAge" > 21) AND ("toAge" < 30))
29	Rows Removed by Filter: 198574
30	-> Index Scan using "PK_advertisement" on advertisement a (cost=0.42..2.67 rows=1 width=24) (actual time=0.002..0.002 rows=0 loops=1...
31	Index Cond: ("advertisementID" = j."advertisementID")
32	Filter: ("datePosted" >= (CURRENT_DATE - '6 mons'::interval))
33	Rows Removed by Filter: 1
34	-> Materialize (cost=0.00..67328.53 rows=1326853 width=35) (actual time=0.011..117.799 rows=1330542 loops=164)
35	-> Seq Scan on education e (cost=0.00..50327.26 rows=1326853 width=35) (actual time=0.090..393.899 rows=1330542 loops=1)
36	Filter: ((country)::text = 'Canada'::text)
37	Rows Removed by Filter: 553479
38	Planning Time: 0.236 ms
39	Execution Time: 35670.028 ms

Total rows: 39 of 39 | Query complete 00:00:36.143

Πάλι το Planning Time δεν έχει αλλάξει ιδιαίτερα, όμως το Execution Time έχει αυξηθεί ακόμα περισσότερο. Ακόμη παρατηρούμε ότι ενώ απενεργοποιήσαμε όλες τις μεθόδους σύνδεσης, το Nested Loop Join χρησιμοποιείται.
Planning Time: 0.236 ms
Execution Time: 35670.028 ms

Έτσι συμπεραίνουμε ότι ούτε στην αρχική μας περίπτωση, χωρίς κάποιο ευρετήριο ή cluster, θα ήταν αποδοτική η απενεργοποίηση κάποιου αλγόριθμου σύνδεσης.
Μία παρατήρηση που κάναμε είναι ότι και στις 2 περιπτώσεις όταν απενεργοποιήσαμε και τους 3 αλγόριθμους, χρησιμοποιείται αναγκαστικά ο Nested loop Join. Αυτό συμβαίνει στην PostgreSQL επειδή το Nested Loop Join είναι ο πιο βασικός και γενικός αλγόριθμος σύνδεσης που μπορεί να χρησιμοποιηθεί ανεξάρτητα από τη δομή των δεδομένων και των ευρετηρίων.

Αλλαγή σειράς των συνδέσεων

Αλλάζουμε τη σειρά των Joins των δύο subqueries που έχουμε στο ερωτημά μας.

```sql
--Allagh seiras syndesewn
EXPLAIN ANALYZE
SELECT e."eduLevel", COUNT(*)
FROM education e
JOIN (
    SELECT m."receiverEmail"
    FROM msg m
    WHERE m."dateSent" >= CURRENT_DATE - INTERVAL '6 months'
    GROUP BY m."receiverEmail"
) AS recent_msgs ON e.email = recent_msgs."receiverEmail"
JOIN (
    SELECT a.email
    FROM advertisement a
    JOIN "jobOffer" j ON a."advertisementID" = j."advertisementID"
    WHERE a."datePosted" >= CURRENT_DATE - INTERVAL '6 months'
    AND j."fromAge" > 21 AND j."toAge" < 30
    GROUP BY a.email
    HAVING COUNT(a."advertisementID") >= 2
) AS valid_ads ON e.email = valid_ads.email
WHERE e.country = 'Canada'
GROUP BY e."eduLevel";
```

Χωρις ευρετήρια/clusters:

38	Planning Time: 0.289 ms
39	Execution Time: 1085.802 ms
Total rows: 39 of 39	Query complete 00:00:01.113
38	Planning Time: 0.355 ms
39	Execution Time: 1104.399 ms
Total rows: 39 of 39	Query complete 00:00:01.139
38	Planning Time: 0.635 ms
39	Execution Time: 1135.770 ms
Total rows: 39 of 39	Query complete 00:00:01.169

Με τα ευρετήρια και clusters που έχουμε δημιουργήσει:

40	Planning Time: 0.436 ms
41	Execution Time: 280.795 ms
Total rows: 41 of 41	Query complete 00:00:00.315
40	Planning Time: 0.399 ms
41	Execution Time: 285.110 ms
Total rows: 41 of 41	Query complete 00:00:00.316
40	Planning Time: 0.491 ms
41	Execution Time: 282.808 ms
Total rows: 41 of 41	Query complete 00:00:00.311

Παρατηρούμε ότι χωρίς τη χρήση ευρετηρίων/clusters και με αλλαγή της σειράς σύνδεσης έχουμε παρόμοιους χρόνους σε σχέση με το αρχικό μας ερώτημα και ίσως λίγο αυξημένους.

Με τη χρήση των ευρετηρίων και των clusters όμως που είδαμε ότι επιταχύνουν το ερώτημα και με αλλαγή της σειράς σύνδεσης έχουμε τους καλύτερους δυνατούς χρόνους εκτέλεσης του ερωτήματός μας.

Επιλογή διαφορετικής χώρας

Κάνοντας χρήση των στατιστικών, βρίσκουμε τη χώρα με τη μικρότερη συχνότητα εμφάνισης στη στήλη country του πίνακα education.

```
ANALYZE education;

SELECT most_common_vals
FROM pg_stats
WHERE tablename = 'education'
AND attname = 'country';
```

Βρίσκουμε ότι η χώρα με τη μικρότερη συχνότητα εμφάνισης είναι το El Salvador. Πράγματι:

	eduLevel character varying (30)	count bigint
1	Bachelor	1
2	Master	1
3	PhD	1
4	Secondary School Degree	1

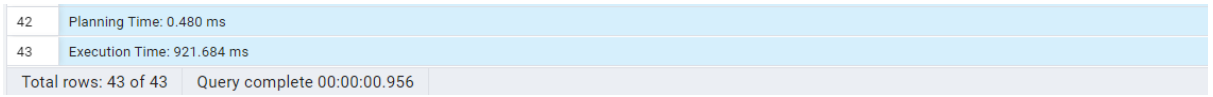
Χωρίς ευρετήρια και clusters:

38	Planning Time: 0.569 ms
39	Execution Time: 876.662 ms
Total rows: 39 of 39	Query complete 00:00:00.911
38	Planning Time: 0.683 ms
39	Execution Time: 982.482 ms
Total rows: 39 of 39	Query complete 00:00:01.012
38	Planning Time: 0.409 ms
39	Execution Time: 921.442 ms
Total rows: 39 of 39	Query complete 00:00:00.952

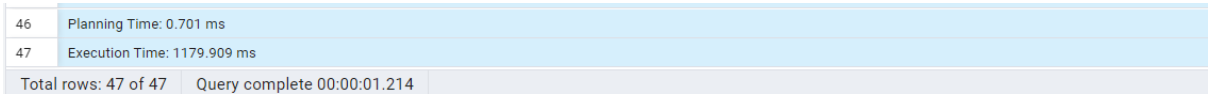
Με ευρετήρια και clusters:

40	Planning Time: 0.587 ms
41	Execution Time: 295.325 ms
Total rows: 41 of 41	Query complete 00:00:00.325
40	Planning Time: 0.364 ms
41	Execution Time: 297.544 ms
Total rows: 41 of 41	Query complete 00:00:00.322
40	Planning Time: 0.604 ms
41	Execution Time: 307.402 ms
Total rows: 41 of 41	Query complete 00:00:00.348

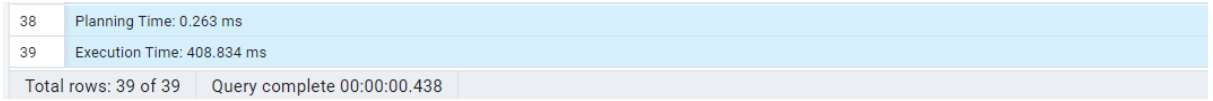
Με ευρετήρια και clusters και hashjoin = off:



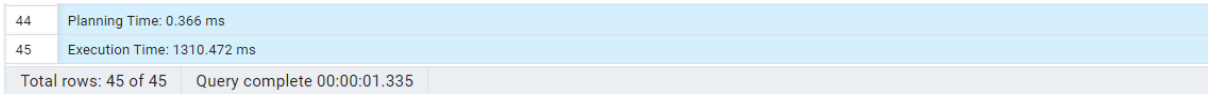
Με ευρετήρια και clusters και hashjoin = off, nestloop = off:



Με ευρετήρια και clusters και hashjoin = off, nestloop = off, mergejoin = off:

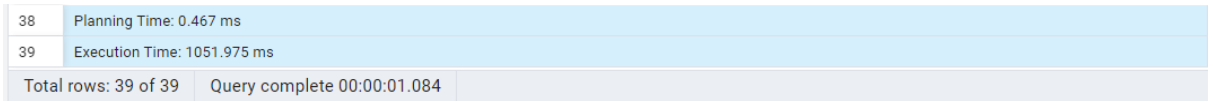


Χωρίς ευρετήρια και clusters και hashjoin = off:

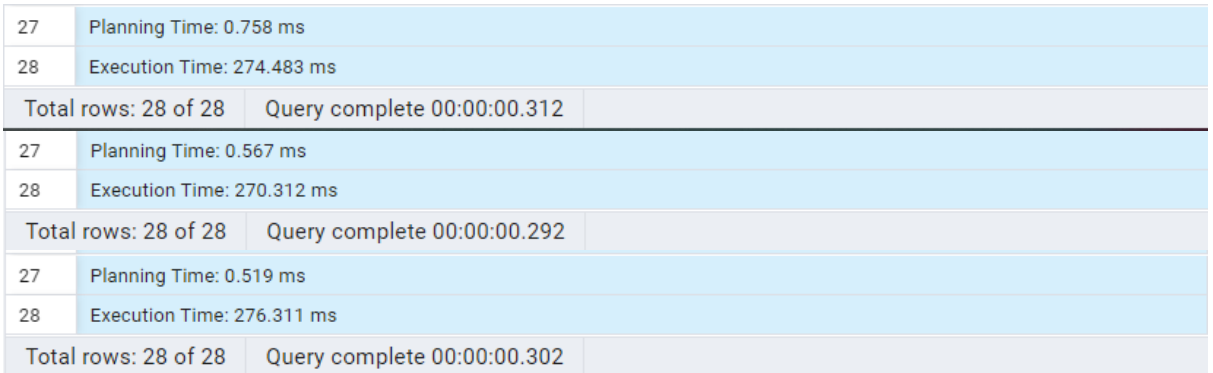


Χωρίς ευρετήρια και clusters και hashjoin = off, mergejoin = off:

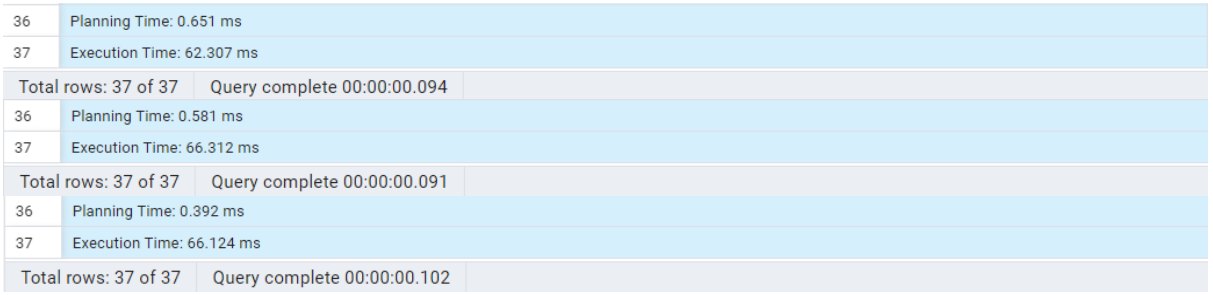
Χωρίς ευρετήρια και clusters και hashjoin = off, mergejoin = off, nestloop = off:



Χωρίς ευρετήρια και clusters και με αλλαγή της σειράς συνδέσεων:



Με ευρετήρια και clusters και με αλλαγή της σειράς συνδέσεων:



Γενικά, η μικρή κατανομή δεδομένων για το Ελ Σαλβαδόρ (σε σχέση με τον Καναδά) οδήγησε σε μικρότερο αριθμό γραμμών που επεξεργάζονται, με αποτέλεσμα το μικρότερο execution time. Και στα δύο πλάνα, χρησιμοποιούνται hash joins και hash aggregates, κάτι που δείχνει ότι είναι οι βέλτιστοι αλγόριθμοι για τα συγκεκριμένα δεδομένα και ερωτήματα. Όσον αφορά το ερώτημα για το Ελ Σαλβαδόρ, έχουμε και εδώ τους καλύτερους δυνατούς χρόνους με χρήση των κατάλληλων ευρετηρίων και clusters καθώς και με την αλλαγή στη σειρά των συνδέσεων.

Συμπέρασμα

Καταλήγουμε, ότι αφού εξετάσαμε όλες τις δυνατές περιπτώσεις και έπειτα από μελέτη με τη χρήση της *EXPLAIN ANALYZE* το αίτημα επιταχύνεται όσο το δυνατόν περισσότερο με τη χρήση ευρετηρίων για τον πίνακα education στο email (hash), για το advertisement στο datePosted (B-Tree), για το msg στο dateSent (B-Tree), με την ομαδοποίηση στο advertisement και στο msg και με την αλλαγή της σειράς των συνδέσεων.