ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ – ΑΝΑΦΟΡΑ ΦΑΣΗΣ Β

Κωνσταντίνος Μιχαλόχρηστας 2020030111
Ευσταθία-Μαρία Χατζηαθανασίου 2019030028

ΖΗΤΟΥΜΕΝΟ 2

Χρησιμοποιώντας την εξής ερώτηση :

```
EXPLAIN ANALYSE select ci.name,count(st.amka) as students
from "Cities" ci join "Person" per on( ci.id=per.city_id) join "Student" st using(amka) join "Joins" jo
on(st.amka=jo."StudentAMKA") join "Program" pro using("ProgramID")
where ci.population > 50000 and (select count("ProgramID") from "Program" where "Duration" =5)>=2 and
st.entry_date>= '2040-09-01'::date and st.entry_date <= '2050-09-30'::date
group by ci.name;
```

Χωρίς καμία χρήση ευρετηρίου έχουμε τα εξής αποτελέσματα :

```
"GroupAggregate  (cost=175859.84..175891.06 rows=17 width=30) (actual time=20833.476..20842.419 rows=6 loops=1)"
"  Group Key: ci.name"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=2.53..2.54 rows=1 width=8) (actual time=0.046..0.047 rows=1 loops=1)"
"          ->  Seq Scan on ""Program""  (cost=0.00..2.50 rows=11 width=4) (actual time=0.034..0.043 rows=11 loops=1)"
"                Filter: (""Duration"" = 5)"
"                Rows Removed by Filter: 29"
"  ->  Sort  (cost=175857.30..175867.65 rows=4140 width=34) (actual time=20802.994..20831.609 rows=87384 loops=1)"
"        Sort Key: ci.name"
"        Sort Method: external merge  Disk: 2984kB"
"        ->  Result  (cost=16.53..175608.59 rows=4140 width=34) (actual time=4235.324..20719.376 rows=87384 loops=1)"
"              One-Time Filter: ($0 >= 2)"
"              ->  Hash Join  (cost=16.53..175608.59 rows=4140 width=34) (actual time=4235.276..20700.685 rows=87384
loops=1)"
"                    Hash Cond: (jo.""ProgramID"" = pro.""ProgramID"")"
"                    ->  Nested Loop  (cost=13.63..175593.66 rows=4140 width=38) (actual time=4235.246..20661.332
rows=87384 loops=1)"
"                          ->  Nested Loop  (cost=13.20..173680.74 rows=4064 width=46) (actual time=4235.179..19323.497
rows=84777 loops=1)"
"                                ->  Hash Join  (cost=12.78..125974.21 rows=73709 width=34) (actual time=0.120..1872.405
rows=1513928 loops=1)"
"                                      Hash Cond: (per.city_id = ci.id)"
"                                      ->  Seq Scan on ""Person"" per  (cost=0.00..120848.41 rows=1929441 width=16) (actual
time=0.039..1244.941 rows=1929453 loops=1)"
"                                      ->  Hash  (cost=12.56..12.56 rows=17 width=26) (actual time=0.073..0.074 rows=19 loops=1)"
"                                            Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                                            ->  Seq Scan on ""Cities"" ci  (cost=0.00..12.56 rows=17 width=26) (actual time=0.008..0.069
rows=19 loops=1)"
"                                                  Filter: (population > '50000'::numeric)"
"                                                  Rows Removed by Filter: 426"
"                                ->  Index Scan using ""Student_pkey"" on ""Student"" st  (cost=0.43..0.65 rows=1 width=12) (actual
time=0.011..0.011 rows=0 loops=1513928)"
"                                      Index Cond: ((amka)::text = (per.amka)::text)"
"                                      Filter: ((entry_date >= '2040-09-01'::date) AND (entry_date <= '2050-09-30'::date))"
"                                      Rows Removed by Filter: 1"
```

```
"                    ->  Index Only Scan using ""Joins_pkey"" on ""Joins"" jo  (cost=0.43..0.46 rows=1 width=16) (actual
time=0.014..0.015 rows=1 loops=84777)"
"                          Index Cond: (""StudentAMKA"" = (per.amka)::text)"
"                          Heap Fetches: 0"
"                ->  Hash  (cost=2.40..2.40 rows=40 width=4) (actual time=0.017..0.017 rows=40 loops=1)"
"                    Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                    ->  Seq Scan on ""Program"" pro  (cost=0.00..2.40 rows=40 width=4) (actual time=0.007..0.010 rows=40
loops=1)"
"Planning Time: 5.888 ms"
```
"Execution Time: 20843.366 ms"

2η εκτέλεση : **"Execution Time: 21107.412 ms"**
3η εκτέλεση : **"Execution Time: 20601.002 ms"**

Γνωρίζοντας πως τα hash indexes είναι καλύτερα για εύρεση συγκεκριμένου σημείου, θα δημιουργήσουμε αντίστοιχα για το Student.amka, Person.amka, Joins.ProgramID, Cities.id, Program.ProgramID, λόγω τον join που κάνουμε ανάμεσα στους πίνακες στην έρωτηση. Θα δημιουργήσουμε, επίσης, ένα hash index για το Program.Duration, διότι θέλουμε Duration =5. Τέλος, θα φτιάξουμε btree indexes για το Cities.population, Student.entry_date, διότι σε αυτά αναζητούμε εύρος (population > 50000, 2040-09-01<entry_date<2050-09-30.

Με τις παραπάνω αλλαγές παίρνουμε το παρακάτω EXPLAIN ANALYSE:

Με: enable_hashjoin = off
enable_mergejoin = off
max_parallel_workers_per_gather = 0

```
"GroupAggregate  (cost=110226.36..110255.48 rows=17 width=30) (actual time=4283.682..4292.078 rows=6 loops=1)"
"  Group Key: ci.name"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=2.53..2.54 rows=1 width=8) (actual time=0.040..0.040 rows=1 loops=1)"
"          ->  Seq Scan on ""Program""  (cost=0.00..2.50 rows=11 width=4) (actual time=0.023..0.027 rows=11 loops=1)"
"                Filter: (""Duration"" = 5)"
"                Rows Removed by Filter: 29"
"  ->  Sort  (cost=110223.82..110233.47 rows=3861 width=34) (actual time=4256.597..4282.329 rows=87384 loops=1)"
"        Sort Key: ci.name"
"        Sort Method: external merge  Disk: 2984kB"
"        ->  Result  (cost=0.86..109993.80 rows=3861 width=34) (actual time=0.234..4188.843 rows=87384 loops=1)"
"              One-Time Filter: ($0 >= 2)"
"              ->  Nested Loop  (cost=0.86..109993.80 rows=3861 width=34) (actual time=0.192..4174.563 rows=87384
loops=1)"
"                    ->  Nested Loop  (cost=0.85..109890.12 rows=3861 width=38) (actual time=0.187..4065.375 rows=87384
loops=1)"
"                          ->  Nested Loop  (cost=0.85..107870.30 rows=101065 width=20) (actual time=0.129..3857.058
rows=111531 loops=1)"
"                                Join Filter: ((st.amka)::text = (per.amka)::text)"
"                                ->  Nested Loop  (cost=0.85..82267.33 rows=101066 width=28) (actual time=0.109..2957.757
rows=111531 loops=1)"
"                                      ->  Index Scan using student_ed_idx on ""Student"" st  (cost=0.43..6841.51 rows=99204 width=12)
(actual time=0.072..1236.891 rows=108262 loops=1)"
"                                            Index Cond: ((entry_date >= '2040-09-01'::date) AND (entry_date <= '2050-09-30'::date))"
```

```
"                              ->  Index Only Scan using ""Joins_pkey"" on ""Joins"" jo  (cost=0.43..0.75 rows=1 width=16)
(actual time=0.015..0.015 rows=1 loops=108262)"
"                                    Index Cond: (""StudentAMKA"" = (st.amka)::text)"
"                                    Heap Fetches: 0"
"                           ->  Index Scan using person_idx on ""Person"" per  (cost=0.00..0.24 rows=1 width=16) (actual
time=0.007..0.007 rows=1 loops=111531)"
"                                 Index Cond: ((amka)::text = (jo.""StudentAMKA"")::text)"
"                                 Rows Removed by Index Recheck: 0"
"                       ->  Index Scan using city_idx on ""Cities"" ci  (cost=0.00..0.02 rows=1 width=26) (actual
time=0.001..0.001 rows=1 loops=111531)"
"                             Index Cond: (id = per.city_id)"
"                             Filter: (population > '50000'::numeric)"
"                             Rows Removed by Filter: 0"
"                  ->  Index Scan using program_idx on ""Program"" pro  (cost=0.00..0.02 rows=1 width=4) (actual
time=0.001..0.001 rows=1 loops=87384)"
"                        Index Cond: (""ProgramID"" = jo.""ProgramID"")"
"Planning Time: 0.730 ms"
```
"**Execution Time: 4292.888 ms**"

2η εκτέλεση : **"Execution Time: 4304.901 ms"**
3η εκτέλεση : **"Execution Time: 4185.761 ms"**

Οπότε παρατηρούμε πως ο τελικός χρόνος πριν το clustering μειώθηκε από **20843.366 ms** σε **4292.888 ms.** Στην επόμενη φάση θα χρησιμοποιήσουμε συσταδοποιήσεις για να μειώσουμε περισσότερο το execution time.

Θα δοκιμάσουμε να κάνουμε πρώτα clustering στα 2 btree indexes που ήδη έχουμε.

Χρησιμοποιώντας clustering στα 2 btree indexes έχουμε :

```
"GroupAggregate  (cost=110226.34..110255.46 rows=17 width=30) (actual time=2345.426..2356.314 rows=6 loops=1)"
"  Group Key: ci.name"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=2.53..2.54 rows=1 width=8) (actual time=0.025..0.026 rows=1 loops=1)"
"          ->  Seq Scan on ""Program""  (cost=0.00..2.50 rows=11 width=4) (actual time=0.019..0.021 rows=11 loops=1)"
"                Filter: (""Duration"" = 5)"
"                Rows Removed by Filter: 29"
"  ->  Sort  (cost=110223.80..110233.45 rows=3861 width=34) (actual time=2317.030..2346.000 rows=87384 loops=1)"
"        Sort Key: ci.name"
"        Sort Method: external merge  Disk: 2984kB"
"        ->  Result  (cost=0.86..109993.78 rows=3861 width=34) (actual time=0.177..2263.244 rows=87384 loops=1)"
"              One-Time Filter: ($0 >= 2)"
"              ->  Nested Loop  (cost=0.86..109993.78 rows=3861 width=34) (actual time=0.150..2249.953 rows=87384
loops=1)"
"                    ->  Nested Loop  (cost=0.85..109890.10 rows=3861 width=38) (actual time=0.132..2150.816 rows=87384
loops=1)"
"                          ->  Nested Loop  (cost=0.85..107870.30 rows=101064 width=20) (actual time=0.126..1967.597
rows=111531 loops=1)"
"                                Join Filter: ((st.amka)::text = (per.amka)::text)"
"                                ->  Nested Loop  (cost=0.85..82267.33 rows=101066 width=28) (actual time=0.108..1378.123
rows=111531 loops=1)"
"                                      ->  Index Scan using student_ed_idx on ""Student"" st  (cost=0.43..6841.51 rows=99204 width=12)
(actual time=0.084..78.802 rows=108262 loops=1)"
```

```
"                                 Index Cond: ((entry_date >= '2040-09-01'::date) AND (entry_date <= '2050-09-30'::date))"
"                               ->  Index Only Scan using ""Joins_pkey"" on ""Joins"" jo  (cost=0.43..0.75 rows=1 width=16)
(actual time=0.011..0.011 rows=1 loops=108262)"
"                                     Index Cond: (""StudentAMKA"" = (st.amka)::text)"
"                                     Heap Fetches: 0"
"                           ->  Index Scan using person_idx on ""Person"" per  (cost=0.00..0.24 rows=1 width=16) (actual
time=0.005..0.005 rows=1 loops=111531)"
"                                 Index Cond: ((amka)::text = (jo.""StudentAMKA"")::text)"
"                                 Rows Removed by Index Recheck: 0"
"                       ->  Index Scan using city_idx on ""Cities"" ci  (cost=0.00..0.02 rows=1 width=26) (actual
time=0.001..0.001 rows=1 loops=111531)"
"                             Index Cond: (id = per.city_id)"
"                             Filter: (population > '50000'::numeric)"
"                             Rows Removed by Filter: 0"
"                   ->  Index Scan using program_idx on ""Program"" pro  (cost=0.00..0.02 rows=1 width=4) (actual
time=0.001..0.001 rows=1 loops=87384)"
"                         Index Cond: (""ProgramID"" = jo.""ProgramID"")"
"Planning Time: 0.637 ms"
"Execution Time: 2357.273 ms"
```

2η εκτέλεση : **"Execution Time: 2389.323 ms"**
3η εκτέλεση : **"Execution Time: 2201.021 ms"**

Παρατηρούμε πως ο χρόνος πέφτει στα **2357.273 ms**.

Έπειτα, σκεφτήκαμε να μετατρέψουμε το index που είχαμε για το Program.Duration σε btree και θα κάναμε cluster ως προς αυτό, όμως το Program έχει μόνο 40 πλειάδες οπότε δεν θα υπήρχε σημαντική αλλαγή στο execution time.

Τώρα θα δοκιμάσουμε τα εξής :

```
set enable_hashjoin=on;
set enable_mergejoin=on;
```

```
"GroupAggregate  (cost=106489.58..106517.96 rows=17 width=30) (actual time=1943.106..1951.034 rows=6 loops=1)"
"  Group Key: ci.name"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=2.53..2.54 rows=1 width=8) (actual time=0.027..0.027 rows=1 loops=1)"
"          ->  Seq Scan on ""Program""  (cost=0.00..2.50 rows=11 width=4) (actual time=0.018..0.022 rows=11 loops=1)"
"                Filter: (""Duration"" = 5)"
"                Rows Removed by Filter: 29"
"  ->  Sort  (cost=106487.04..106496.45 rows=3762 width=34) (actual time=1917.886..1941.226 rows=87384 loops=1)"
"        Sort Key: ci.name"
"        Sort Method: external merge  Disk: 2984kB"
"        ->  Result  (cost=9.64..106263.63 rows=3762 width=34) (actual time=0.270..1862.885 rows=87384 loops=1)"
"              One-Time Filter: ($0 >= 2)"
"              ->  Nested Loop  (cost=9.64..106263.63 rows=3762 width=34) (actual time=0.228..1853.141 rows=87384
loops=1)"
"                    ->  Hash Join  (cost=9.64..106162.59 rows=3762 width=38) (actual time=0.223..1756.974 rows=87384
loops=1)"
"                          Hash Cond: (per.city_id = ci.id)"
```

```
"                    ->  Nested Loop  (cost=0.85..105892.81 rows=98488 width=20) (actual time=0.156..1726.092
rows=111531 loops=1)"
"                          Join Filter: ((st.amka)::text = (per.amka)::text)"
"                          ->  Nested Loop  (cost=0.85..80942.58 rows=98488 width=28) (actual time=0.132..1230.114
rows=111531 loops=1)"
"                                ->  Index Scan using student_ed_idx on ""Student"" st  (cost=0.43..6667.91 rows=96674 width=12)
(actual time=0.111..66.032 rows=108262 loops=1)"
"                                      Index Cond: ((entry_date >= '2040-09-01'::date) AND (entry_date <= '2050-09-30'::date))"
"                                ->  Index Only Scan using ""Joins_pkey"" on ""Joins"" jo  (cost=0.43..0.76 rows=1 width=16)
(actual time=0.010..0.010 rows=1 loops=108262)"
"                                      Index Cond: (""StudentAMKA"" = (st.amka)::text)"
"                                      Heap Fetches: 0"
"                          ->  Index Scan using person_idx on ""Person"" per  (cost=0.00..0.24 rows=1 width=16) (actual
time=0.004..0.004 rows=1 loops=111531)"
"                                Index Cond: ((amka)::text = (jo.""StudentAMKA"")::text)"
"                                Rows Removed by Index Recheck: 0"
"                    ->  Hash  (cost=8.57..8.57 rows=17 width=26) (actual time=0.036..0.036 rows=19 loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                          ->  Index Scan using city_pop_idx on ""Cities"" ci  (cost=0.27..8.57 rows=17 width=26) (actual
time=0.027..0.030 rows=19 loops=1)"
"                                Index Cond: (population > '50000'::numeric)"
"               ->  Index Scan using program_idx on ""Program"" pro  (cost=0.00..0.02 rows=1 width=4) (actual
time=0.001..0.001 rows=1 loops=87384)"
"                     Index Cond: (""ProgramID"" = jo.""ProgramID"")"
"Planning Time: 1.841 ms"
```

"**Execution Time: 1952.260 ms**"

2η εκτέλεση : **Execution Time: 2221.791 ms**
3η εκτέλεση : **Execution Time: 1981.047 ms**

Παρατηρούμε πως ο χρόνος μειώνεται ακόμα περισσότερο στα **1952.260 ms.**

Κάνοντας διάφορες αλλάγες, διαγράφοντας indexes, αλλάζοντας τη σειρά των joins στο select. Τα αποτελέσματα μας δείχνουν πως ο λιγότερος χρόνος μπορεί να επιτευχθεί με τις παραπάνω αλλαγές. Επίσης, αν θέσουμε : SET max_parallel_workers_per_gather = 2;

```
"Finalize GroupAggregate  (cost=65724.35..65743.21 rows=17 width=30) (actual time=977.685..1030.053 rows=6
loops=1)"
"  Group Key: ci.name"
"  InitPlan 1 (returns $0)"
"    ->  Aggregate  (cost=2.53..2.54 rows=1 width=8) (actual time=0.060..0.061 rows=1 loops=1)"
"          ->  Seq Scan on ""Program""  (cost=0.00..2.50 rows=11 width=4) (actual time=0.046..0.058 rows=11 loops=1)"
"                Filter: (""Duration"" = 5)"
"                Rows Removed by Filter: 29"
"  ->  Gather Merge  (cost=65721.81..65740.42 rows=17 width=30) (actual time=977.424..1030.039 rows=12 loops=1)"
"        Workers Planned: 1"
"        Params Evaluated: $0"
"        Workers Launched: 1"
"        ->  Partial GroupAggregate  (cost=64721.80..64738.50 rows=17 width=30) (actual time=944.718..948.583 rows=6
loops=2)"
"              Group Key: ci.name"
"              ->  Sort  (cost=64721.80..64727.31 rows=2204 width=34) (actual time=937.191..943.817 rows=43692 loops=2)"
"                    Sort Key: ci.name"
"                    Sort Method: external merge  Disk: 1584kB"
"                    Worker 0:  Sort Method: external merge  Disk: 1408kB"
"                    ->  Result  (cost=12.54..64599.41 rows=2204 width=34) (actual time=1.115..913.112 rows=43692 loops=2)"
```

```
"                  One-Time Filter: ($0 >= 2)"
"                  ->  Hash Join  (cost=12.54..64599.41 rows=2204 width=34) (actual time=1.114..908.715 rows=43692
loops=2)"
"                        Hash Cond: (jo.""ProgramID"" = pro.""ProgramID"")"
"                        ->  Hash Join  (cost=9.64..64590.11 rows=2204 width=38) (actual time=0.881..898.998 rows=43692
loops=2)"
"                              Hash Cond: (per.city_id = ci.id)"
"                              ->  Nested Loop  (cost=0.85..64428.49 rows=57672 width=20) (actual time=0.644..884.967
rows=55766 loops=2)"
"                                    Join Filter: ((st.amka)::text = (per.amka)::text)"
"                                    ->  Nested Loop  (cost=0.85..49817.78 rows=57674 width=28) (actual time=0.489..609.286
rows=55766 loops=2)"
"                                          ->  Parallel Index Scan using student_ed_idx on ""Student"" st  (cost=0.43..6242.96
rows=56612 width=12) (actual time=0.184..33.275 rows=54131 loops=2)"
"                                                Index Cond: ((entry_date >= '2040-09-01'::date) AND (entry_date <= '2050-09-
30'::date))"
"                                          ->  Index Only Scan using ""Joins_pkey"" on ""Joins"" jo  (cost=0.43..0.76 rows=1
width=16) (actual time=0.010..0.010 rows=1 loops=108262)"
"                                                Index Cond: (""StudentAMKA"" = (st.amka)::text)"
"                                                Heap Fetches: 0"
"                                    ->  Index Scan using person_idx on ""Person"" per  (cost=0.00..0.24 rows=1 width=16) (actual
time=0.004..0.004 rows=1 loops=111531)"
"                                          Index Cond: ((amka)::text = (jo.""StudentAMKA"")::text)"
"                                          Rows Removed by Index Recheck: 0"
"                              ->  Hash  (cost=8.57..8.57 rows=17 width=26) (actual time=0.163..0.164 rows=19 loops=2)"
"                                    Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                                    ->  Index Scan using city_pop_idx on ""Cities"" ci  (cost=0.27..8.57 rows=17 width=26) (actual
time=0.152..0.155 rows=19 loops=2)"
"                                          Index Cond: (population > '50000'::numeric)"
"                        ->  Hash  (cost=2.40..2.40 rows=40 width=4) (actual time=0.190..0.191 rows=40 loops=2)"
"                              Buckets: 1024  Batches: 1  Memory Usage: 10kB"
"                              ->  Seq Scan on ""Program"" pro  (cost=0.00..2.40 rows=40 width=4) (actual time=0.170..0.179
rows=40 loops=2)"
"Planning Time: 2.010 ms"
```

"Execution Time: 1030.969 ms"

2η εκτέλεση : **Execution Time: 1130.096 ms**
3η εκτέλεση : **Execution Time: 1281.075 ms**

Παρατηρούμε πως ο χρόνος εκτέλεσης είναι ακόμα μικρότερος.

ΖΗΤΟΥΜΕΝΟ 3

Θέτουμε :

SET max_parallel_workers_per_gather = 0;
set enable_hashjoin=off;
set enable_mergejoin=off;

Αρχικά, θα τρέξουμε την συνάρτηση insert_person() για1.000 άτομα.

"Function Scan on insert_person (cost=0.25..10.25 rows=1000 width=96) (actual time=233948.826..233948.863 rows=1000 loops=1)"
"Planning Time: 0.031 ms"
"Execution Time: 233948.940 ms"

2η εκτέλεση : **Execution Time: 233438.586 ms**
3η εκτέλεση : **Execution Time: 235288.271 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

"Function Scan on insert_person (cost=0.25..10.25 rows=1000 width=96) (actual time=219521.449..219521.487 rows=998 loops=1)"
"Planning Time: 0.033 ms"
"Execution Time: 219521.577 ms"

2η εκτέλεση : **Execution Time: 215627.298 ms**
3η εκτέλεση : **Execution Time: 238800.296 ms**

--Με χρήση hash index για το amka

και set enable_hashjoin=off;
set enable_mergejoin=off;

2η εκτέλεση : **Execution Time: 232334.333 ms**
3η εκτέλεση : **Execution Time: 229310.301 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 267213.277 ms**
2η εκτέλεση : **Execution Time: 227711.160 ms**
3η εκτέλεση : **Execution Time: 235116.182 ms**

Ξαναθέτουμε

set enable_hashjoin=off;
set enable_mergejoin=off;

και τρέχουμε την insert_student() για 100 μαθητες

"Function Scan on insert_student (cost=0.25..10.25 rows=1000 width=132) (actual time=224151.107..224151.108 rows=0 loops=1)"
"Planning Time: 0.020 ms"
"Execution Time: 224151.123 ms"

2η εκτέλεση : **Execution Time: 226922.484 ms**
3η εκτέλεση : **Execution Time: 199856.192 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 208022.702 ms**
2η εκτέλεση : **Execution Time : 200952.881 ms**
3η εκτέλεση : **Execution Time: 199055.273 ms**

Με χρήση hash index στο amka

set enable_hashjoin=off;
set enable_mergejoin=off;

1η εκτέλεση : **Execution Time: 186755.621 ms**
2η εκτέλεση : **Execution Time: 188357.457 ms**
3η εκτέλεση : **Execution Time: 188955.091 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 187807.085 ms**
2η εκτέλεση : **Execution Time: 194010.272 ms**
3η εκτέλεση : **Execution Time: 194109.399 ms**

Προχωράμε στη συνάρτηση insert_students_to_programs() για τον πίνακα Joins. Δοκιμάζουμε εισαγωγή 1000 εγγραφών, χωρίς κάποιο ευρετήριο.
set enable_hashjoin=off;
set enable_mergejoin=off;

1η εκτέλεση : **Execution Time: 296460.463 ms**
2η εκτέλεση : **Execution Time: 301514.316 ms**
3η εκτέλεση : **Execution Time: 302782.258 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 294858.171 ms**
2η εκτέλεση : **Execution Time: 301366.472 ms**
3η εκτέλεση : **Execution Time: 301763.971 ms**

Με χρήση hash index στο StudentAMKA και στο ProgramID

set enable_hashjoin=off;
set enable_mergejoin=off;

1η εκτέλεση : **Execution Time: 291022.452 ms**
2η εκτέλεση : **Execution Time: 302982.777 ms**
3η εκτέλεση : **Execution Time: 304040.455 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 297150.746 ms**
2η εκτέλεση : **Execution Time: 301953.213 ms**
3η εκτέλεση : **Execution Time: 302951.537 ms**

Τέλος πηγαίνουμε στον πίνακα “Program” με την συνάρτηση insert_program() για 100.000 εισαγωγές
Θέτουμε
set enable_hashjoin=off;
set enable_mergejoin=off;

1η εκτέλεση : **Execution Time: 4846.344 ms**
2η εκτέλεση : **Execution Time: 4909.202 ms**
3η εκτέλεση : **Execution Time: 5931.231 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 3077.395 ms**
2η εκτέλεση : **Execution Time: 3133.766 ms**
3η εκτέλεση : **Execution Time: 3215.559 ms**

Με χρήση hash στο ProgramID

set enable_hashjoin=off;
set enable_mergejoin=off;

1η εκτέλεση : **Execution Time: 5232.663 ms**
2η εκτέλεση : **Execution Time: 5888.712 ms**
3η εκτέλεση : **Execution Time: 7921.567 ms**

Παρατηρούμε πως οι χρόνοι εκτέλεσης γίνονται μεγαλύτεροι, οπότε θα χρησιμοποιήσουμε btree

1η εκτέλεση : **Execution Time: 3169.300 ms**
2η εκτέλεση : **Execution Time: 3405.641 ms**
3η εκτέλεση : **Execution Time: 3386.209 ms**

set enable_hashjoin=on;
set enable_mergejoin=on;

1η εκτέλεση : **Execution Time: 3143.910 ms**
2η εκτέλεση : **Execution Time: 3037.507 ms**
3η εκτέλεση : **Execution Time: 3364.818 ms**

Συνοπτικά παρατηρούμε πως τα ευρετήρια που χρησιμοποιήσαμε δεν δημιουργούν σημαντική διαφορά χρόνου εκτέλεσης σε **καμία** εισαγωγή, το μόνο index που έκανε διαφορά ήταν στην περίπτωση του hash στο ProgramID, οπού αύξησε τον χρόνο εκτέλεσης.