Β Α Σ Ε Ι Σ Δ Ε Δ Ο Μ Ε Ν Ω Ν

(Π Λ Η302)

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΩΝ ΕΡΓΑΣΙΩΝ ΤΟΥ ΜΑΘΗΜΑΤΟΣ

Ομάδα Χρηστών 190

Γουρνέλος Αλέξιος : 2016030091

Κατσίμπας Πέτρος : 2016030038

Συνοπτική Περιγραφή Ά Φάσης

Ζητούμενο 1:

Επιπλέον από τους πίνακες που μας δόθηκαν στην αρχική βάση δεδομένων δημιουργήθηκε και ο πίνακας Activity,Participates,Manages και Transaction σύμφωνα με το διάγραμμα που μας δόθηκε.Επιπλέον,για δικία μας ευκολία δημιουργήσαμε τις συναρτήσεις getnextid() όπου υπολογίζει το μέγιστο idPerson,και μας βοηθήσε στο να ορίσουμε το επόμενο id στην insert() καθως και την συνάρτηση getnexttran() όπου κάνει την ίδια αριβώς δουλειά γιια το idTransaction.

Ζητούμενο 2:

- *Ερώτημα 1*: **"2_1_insert_update"**(act character varying,
  documentcli character varying,
  fnam character varying,
  lnam character varying,
  s character,
  birth date,
  addr character varying,
  cit character varying,
  countr character varying,
  cardtyper character varying,
  cardnumber character varying,
  cardholder character varying,
  expirationdate date)
- *Ερώτημα 2:* "2_2_InsertRoomBooking_In_Specific_Range_of_Time"(
  ins_idhotel integer,
  starting_date date,
  ending_date date)

Ζητούμενο 3:

- *Ερώτημα 1:* "3_1_CountryCity_Discount_over30"()
- *Ερώτημα 2*: "3_2_Hotel_Stars_Studio_MinPrice"(hotelstars character varying,
  sletter character varying)

- *Ερώτημα 3*: "3_3_Hotel_RoomTypes_best_Discount"()
- *Ερώτημα 4*: "3_4_Reservation_of_Hotel_ID"(hotelid integer)
- *Ερώτημα 5*: "3_5_Activities_Without_Any_Participant"(idofhotel integer)
- *Ερώτημα 6*: "3_6_Show_Facilities_Subtypes"(typeoffacility character varying)

- *Ερώτημα 7*: "3_7_printHotelnRoom_With_Wanted_Facilities"(
  wantedfacility character varying,
  wantedroomfacility character varying)
- *Ερώτημα 8:* "3_8_printHotels_With_FreeRoomOfAllTypes"()

***Ζητούμενο 4*:**

- *Ερώτημα 1:* "4_1_activitycount_person_parictipates"(idofhotel integer,
  idofperson integer)
- *Ερώτημα 2*: "4_2_Average_Age_to_RoomTypes"()
- *Ερώτημα 3:* "4_3_BestPrice_based_RoomType_in_Country"(
  chosencountry character varying)
- *Ερώτημα 4:* "4_4_track_hotels_over_average_city_income"()
- *Ερώτημα 5 :* "4_5_occupancy_of_hotel_byYear"(
  shotel integer,
  year integer)

***Ζητούμενο 5*:**

- *Ερώτημα 1*: trig5_1_transactionsrecord()
- *Ερώτημα 2:* trig5_2_change_permision_on_roombooking()
  trig5_2_update_permision_on_hotelbooking()
- *Ερώτημα 3:* trig5_3_roombooking_changes_totalamount_dates()

***Ζητούμενο 6*:**

- *Ερωτημα 1:* view6_1() + (viewfunc την επιλογή του ID ξενοδοχείου)
- *Ερωτημα 2:* view6_2 + (viewfunc την επιλογή του ID ξενοδοχείου)

Περιγραφή Β΄ Φάσης

1ο Μέρος

Για το 1ο μέρος της 2ης φάσης με την εφαρμογή επικοινωνίας σε γλώσσα προγραμματισμού JAVA με το σύστημα διαχείρισης βάσεων δεδομένων PostgreSQL μέσω JDBC έχουμε αποστείλει το .zip αρχείο, το οποίο τρέχει όλες τις λειτουργίες που ζητά η εκφώνηση. Βέβαια, για να το πετύχουμε αυτό αναγκαστήκαμε σε *Queries* τύπου ***Insert / Update*** να απενεργοποιήσουμε όλα τα *triggers* ώστε να ενημερώνονται οι πίνακες.

2ο Μέρος

Σε αυτή τη φάση της εργαστηριακής εργασίας εξετάστηκε η απόδοση του ερωτήματος:

"Ταξινομήστε (σε αύξουσα σειρά) ανά μήνα τις συνολικές εισπράξεις για ένα συγκεκριμένο χρονικό διάστημα κρατήσεων δωματίων και για έναν συγκεκριμένο τύπο δωματίων."

Το ερώτημα (***Query***) σε PostgreSQL έχει ως εξής:

```
EXPLAIN ANALYZE
SELECT foo.monthn , CAST(SUM(foo.amount) AS FLOAT) AS am
FROM(
    SELECT hb.totalamount AS amount, EXTRACT('month'FROM rb."checkin") AS monthn
        FROM "hotelbooking" hb
        INNER JOIN "roombooking" rb ON rb."hotelbookingID" = hb."idhotelbooking"
        INNER JOIN "room" r ON r."idRoom" = rb."roomID"
        WHERE(  hb.payed = 'true'AND rb."checkin" > '2021-05-01' AND rb."checkin" < '2021-10-01' AND r."roomtype" = 'Single'   )
    GROUP BY hb."idhotelbooking",rb."roomID",r."roomtype",monthn,r."idHotel"
) AS foo
GROUP BY foo.monthn
ORDER BY foo.monthn
```

Επιλέξαμε το χρονικό διάστημα *'2021-05-01'* έως *'2021-10-01'* και για τύπο δωματίου *'Single'* και είχαμε αυτά τα αποτελέσματα:

	monthn double precision	am double precision
1	5	4249721
2	6	4362159
3	7	4324582
4	8	4798284
5	9	4502404
6	10	2109756

Παρακάτω παρατίθενται τα αποτελέσματα και ο σχολιασμός των αποτελεσμάτων για τα διάφορα EXPLAIN ANALYSE που πραγματοποιήθηκαν στο παραπάνω query, με ή χωρίς indexes, και με ή χωρίς clustering.

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query χωρίς κάποιο Index έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=435032.47..435158.68 rows=200 width=16) (actual time=2318.494..2320.488 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=435032.47..435073.71 rows=16494 width=12) (actual time=2317.974..2318.460 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=433382.28..433877.10 rows=16494 width=12) (actual time=2308.901..2315.719 rows=16011 loops=1)
7	-> Group (cost=433382.28..433712.16 rows=16494 width=32) (actual time=2308.901..2314.771 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=433382.28..433423.51 rows=16494 width=32) (actual time=2308.899..2309.327 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=41.56..432226.90 rows=16494 width=32) (actual time=3.511..2300.952 rows=16011 loops=1)
13	-> Hash Join (cost=41.13..389891.18 rows=33207 width=24) (actual time=0.302..2075.391 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Seq Scan on roombooking rb (cost=0.00..388553.84 rows=492770 width=12) (actual time=0.011..2036.009 rows=484091 lo...
16	Filter: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Rows Removed by Filter: 17698564
18	-> Hash (cost=39.41..39.41 rows=137 width=16) (actual time=0.271..0.272 rows=137 loops=1)
19	Buckets: 1024 Batches: 1 Memory Usage: 15kB
20	-> Seq Scan on room r (cost=0.00..39.41 rows=137 width=16) (actual time=0.007..0.253 rows=137 loops=1)
21	Filter: ((roomtype)::text = 'Single'::text)
22	Rows Removed by Filter: 1896
23	-> Index Scan using "idhotelbooking_PK" on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.006..0.006 rows=0 l...
24	Index Cond: (idhotelbooking = rb."hotelbookingID")
25	Filter: payed
26	Rows Removed by Filter: 1
27	Planning Time: 0.512 ms
28	Execution Time: 2320.770 ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης για το query αυτό εκτιμάται για 0.512 ms (***Planning Time***), ενώ ο χρόνος εκτέλεσης εν τέλει υπολογίζεται στα 2320,770 ms (***Execution Time***).

Έχοντας στο query, συνθήκες τύπου μεγαλύτερου- μικρότερου (**WHERE rb.checkin > '2021-05-01' AND rb.checkin < '2021-10-01'**) αποφασίσαμε, αρχικά, να δημιουργήσουμε ένα ***index*** τύπου b+ tree στον πίνακα ***roombooking*** βασισμένο στο "**checkin**".

--Κώδικας

```
CREATE INDEX checkin_date_idx ON "roombooking" (checkin)
```

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=370415.11..370541.31 rows=200 width=16) (actual time=597.297..599.248 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=370415.11..370456.34 rows=16494 width=12) (actual time=596.704..597.290 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=368764.91..369259.73 rows=16494 width=12) (actual time=587.509..594.346 rows=16011 loops=1)
7	-> Group (cost=368764.91..369094.79 rows=16494 width=32) (actual time=587.508..593.384 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=368764.91..368806.15 rows=16494 width=32) (actual time=587.507..587.938 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10500.89..367609.54 rows=16494 width=32) (actual time=34.557..580.823 rows=16011 loops=1)
13	-> Hash Join (cost=10500.45..325273.81 rows=33207 width=24) (actual time=32.235..335.468 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=31.945..291.640 r...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=28110
18	-> Bitmap Index Scan on btree_checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=28.718..28.718...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=39.41..39.41 rows=137 width=16) (actual time=0.267..0.267 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Seq Scan on room r (cost=0.00..39.41 rows=137 width=16) (actual time=0.009..0.250 rows=137 loops=1)
23	Filter: ((roomtype)::text = 'Single'::text)
24	Rows Removed by Filter: 1896
25	-> Index Scan using "idhotelbooking_PK" on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.007..0.007 rows=0 l...
26	Index Cond: (idhotelbooking = rb."hotelbookingID")
27	Filter: payed
28	Rows Removed by Filter: 1
29	Planning Time: 0.741 ms
30	Execution Time: 599.821 ms

Παρατηρούμε μία αύξηση στο Planning time της τάξης 0.2 ms. Έχουμε όμως μία εμφανή βελτίωση στο Execution time της τάξης 1800 περίπου ms, όπως περιμέναμε.

Το επόμενο μας βήμα ήταν να δοκιμάσουμε **Hash *index*** πάλι στον πίνακα ***roombooking*** χρησιμοποιώντας το column " **checkin** " με σκοπό εύρεσης μίας καλύτερης απόδοσης.

--Κώδικας

```
CREATE INDEX checkin_date_idx ON "roombooking"  USING HASH (checkin)
```

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=370415.11..370541.31 rows=200 width=16) (actual time=742.939..744.923 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=370415.11..370456.34 rows=16494 width=12) (actual time=742.396..742.915 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=368764.91..369259.73 rows=16494 width=12) (actual time=733.462..740.291 rows=16011 loops=1)
7	-> Group (cost=368764.91..369094.79 rows=16494 width=32) (actual time=733.462..739.317 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=368764.91..368806.15 rows=16494 width=32) (actual time=733.460..733.914 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10500.89..367609.54 rows=16494 width=32) (actual time=36.139..726.629 rows=16011 loops=1)
13	-> Hash Join (cost=10500.45..325273.81 rows=33207 width=24) (actual time=33.688..415.432 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=33.392..369.457 r...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=28110
18	-> Bitmap Index Scan on checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=30.023..30.023 rows...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=39.41..39.41 rows=137 width=16) (actual time=0.273..0.273 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Seq Scan on room r (cost=0.00..39.41 rows=137 width=16) (actual time=0.009..0.255 rows=137 loops=1)
23	Filter: ((roomtype)::text = 'Single'::text)
24	Rows Removed by Filter: 1896
25	-> Index Scan using "idhotelbooking_PK" on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.009..0.009 rows=0 l...
26	Index Cond: (idhotelbooking = rb."hotelbookingID")
27	Filter: payed
28	Rows Removed by Filter: 1
29	Planning Time: 1.045 ms
30	Execution Time: 745.531 ms

Δυστυχώς, όμως, δεν παρατηρήσαμε καμία βελτίωση από άποψη χρόνου.

Έπειτα, δοκιμάσαμε να προσθέσουμε **B+- Tree Index** για την αναγκαία σύνδεση μεταξύ των πινάκων ***roombooking, hotelbooking*** στα *ID* όπως φαίνεται στο παρακάτω κώδικα.

```
--Κώδικας

CREATE INDEX hotelbooking_ID_idx ON "roombooking" ("hotelbookingID")

CREATE INDEX ID_hotelbooking_idx ON "hotelbooking" (idhotelbooking)
```

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=370396.75..370522.95 rows=200 width=16) (actual time=581.664..583.592 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=370396.75..370437.98 rows=16494 width=12) (actual time=581.099..581.639 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=368746.55..369241.37 rows=16494 width=12) (actual time=572.223..579.000 rows=16011 loops=1)
7	-> Group (cost=368746.55..369076.43 rows=16494 width=32) (actual time=572.223..578.053 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=368746.55..368787.79 rows=16494 width=32) (actual time=572.221..572.663 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10482.53..367591.18 rows=16494 width=32) (actual time=35.086..565.784 rows=16011 loops=1)
13	-> Hash Join (cost=10482.09..325255.45 rows=33207 width=24) (actual time=32.673..327.951 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=32.509..285.574 r...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=28110
18	-> Bitmap Index Scan on checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=29.155..29.155 rows...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=21.05..21.05 rows=137 width=16) (actual time=0.137..0.137 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Bitmap Heap Scan on room r (cost=5.34..21.05 rows=137 width=16) (actual time=0.038..0.118 rows=137 loops=1)
23	Recheck Cond: ((roomtype)::text = 'Single'::text)
24	Heap Blocks: exact=14
25	-> Bitmap Index Scan on roomtype_date_idx (cost=0.00..5.30 rows=137 width=0) (actual time=0.034..0.034 rows=137 l...
26	Index Cond: ((roomtype)::text = 'Single'::text)
27	-> Index Scan using id_hotelbooking_idx on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.007..0.007 rows=0 l...
28	Index Cond: (idhotelbooking = rb."hotelbookingID")
29	Filter: payed
30	Rows Removed by Filter: 1
31	Planning Time: 1.049 ms
32	Execution Time: 584.168 ms

Παρατηρήσαμε μία ακόμη βελτίωση των 10ms από την προηγούμενη καλύτερη απόδοση καθώς το Index Scan στα ID’s για την συνθήκη ισότητας απέδωσε.

Για άλλη μία φορά, δοκιμάσαμε να προσθέσουμε **Hash Index** αντί για **B+- Tree** στο *ID_attritubes* μεταξύ των πινάκων ***roombooking, hotelbooking.***

```
--Κώδικας

CREATE INDEX hotelbooking_ID_hash_idx ON "roombooking" USING HASH("hotelbookingID")

CREATE INDEX ID_hotelbooking_hash_idx ON "hotelbooking" USING HASH(idhotelbooking)
```

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=357388.69..357514.90 rows=200 width=16) (actual time=801.496..803.478 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=357388.69..357429.93 rows=16494 width=12) (actual time=800.807..801.387 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=355738.50..356233.32 rows=16494 width=12) (actual time=791.970..798.715 rows=16011 loops=1)
7	-> Group (cost=355738.50..356068.38 rows=16494 width=32) (actual time=791.970..797.749 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=355738.50..355779.73 rows=16494 width=32) (actual time=791.968..792.410 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10482.09..354583.12 rows=16494 width=32) (actual time=38.711..784.344 rows=16011 loops=1)
13	-> Hash Join (cost=10482.09..325255.45 rows=33207 width=24) (actual time=32.813..334.150 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=32.553..289.635 r...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=28110
18	-> Bitmap Index Scan on checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=29.311..29.311 rows...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=21.05..21.05 rows=137 width=16) (actual time=0.239..0.239 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Bitmap Heap Scan on room r (cost=5.34..21.05 rows=137 width=16) (actual time=0.069..0.200 rows=137 loops=1)
23	Recheck Cond: ((roomtype)::text = 'Single'::text)
24	Heap Blocks: exact=14
25	-> Bitmap Index Scan on roomtype_date_idx (cost=0.00..5.30 rows=137 width=0) (actual time=0.062..0.063 rows=137 l...
26	Index Cond: ((roomtype)::text = 'Single'::text)
27	-> Index Scan using id_hotelbooking_hash_idx on hotelbooking hb (cost=0.00..0.88 rows=1 width=8) (actual time=0.013..0.013 ro...
28	Index Cond: (idhotelbooking = rb."hotelbookingID")
29	Rows Removed by Index Recheck: 0
30	Filter: payed
31	Rows Removed by Filter: 1
32	Planning Time: 1.665 ms
33	Execution Time: 804.013 ms

Τέλος , προσθέσαμε τα παρακάτω Indexes ώστε να αποφανθούμε στο αποδοτικότερο ευρετήριο για το παραπάνω Query.

--Κώδικας

```
CREATE INDEX roomtype_roomID ON "room"("idRoom") WHERE roomtype = 'Single'
```

(Η απόδοση δεν μεταβλήθηκε αισθητά.)

--Κώδικας

```
CREATE INDEX checkin_date_idx ON "roombooking" (checkin)

CLUSTER "roombooking" USING checkin_date_idx
```

(Αισθητά βελτιωμένη απόδοση με Execution Time = 492.513 ms)

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

Query Editor | Query History | Data Output

	QUERY PLAN text
1	GroupAggregate (cost=370389.90..370516.10 rows=200 width=16) (actual time=490.222..492.158 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=370389.90..370431.13 rows=16494 width=12) (actual time=489.489..490.110 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=368739.70..369234.52 rows=16494 width=12) (actual time=479.624..486.967 rows=16011 loops=1)
7	-> Group (cost=368739.70..369069.58 rows=16494 width=32) (actual time=479.623..485.942 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=368739.70..368780.93 rows=16494 width=32) (actual time=479.621..480.224 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10475.68..367584.32 rows=16494 width=32) (actual time=25.810..472.341 rows=16011 loops=1)
13	-> Hash Join (cost=10475.24..325248.60 rows=33207 width=24) (actual time=25.712..118.440 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=25.590..76.708 ro...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=3084
18	-> Bitmap Index Scan on checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=25.307..25.307 rows...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=14.20..14.20 rows=137 width=16) (actual time=0.111..0.111 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Index Scan using roomtype_roomid on room r (cost=0.14..14.20 rows=137 width=16) (actual time=0.013..0.091 rows=13...
23	-> Index Scan using id_hotelbooking_idx on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.010..0.010 rows=0 l...
24	Index Cond: (idhotelbooking = rb."hotelbookingID")
25	Filter: payed
26	Rows Removed by Filter: 1
27	Planning Time: 1.354 ms
28	Execution Time: 492.513 ms

Επιπρόσθετο Index: *(Το οποίο δοκιμάσαμε και μας έφερε γρηγορότερη εκτέλεση.)*

--Κώδικας

```
CREATE INDEX totalamount_idx ON "hotelbooking"("totalamount") WHERE payed = 'true'
```

Τα αποτελέσματα της ***EXPLAIN ANALYSE*** στο query έχουν ως εξής:

	QUERY PLAN text
1	GroupAggregate (cost=370389.90..370516.10 rows=200 width=16) (actual time=454.238..456.120 rows=5 loops=1)
2	Group Key: foo.monthn
3	-> Sort (cost=370389.90..370431.13 rows=16494 width=12) (actual time=453.668..454.209 rows=16011 loops=1)
4	Sort Key: foo.monthn
5	Sort Method: quicksort Memory: 1135kB
6	-> Subquery Scan on foo (cost=368739.70..369234.52 rows=16494 width=12) (actual time=444.034..451.458 rows=16011 loops=1)
7	-> Group (cost=368739.70..369069.58 rows=16494 width=32) (actual time=444.034..450.387 rows=16011 loops=1)
8	Group Key: hb.idhotelbooking, rb."roomID", r.roomtype, (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
9	-> Sort (cost=368739.70..368780.93 rows=16494 width=32) (actual time=444.033..444.707 rows=16011 loops=1)
10	Sort Key: hb.idhotelbooking, rb."roomID", (date_part('month'::text, (rb.checkin)::timestamp without time zone)), r."idHotel"
11	Sort Method: quicksort Memory: 1635kB
12	-> Nested Loop (cost=10475.68..367584.32 rows=16494 width=32) (actual time=26.818..437.060 rows=16011 loops=1)
13	-> Hash Join (cost=10475.24..325248.60 rows=33207 width=24) (actual time=26.765..118.198 rows=32573 loops=1)
14	Hash Cond: (rb."roomID" = r."idRoom")
15	-> Bitmap Heap Scan on roombooking rb (cost=10459.33..323936.48 rows=492770 width=12) (actual time=26.634..77.276 ro...
16	Recheck Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
17	Heap Blocks: exact=3084
18	-> Bitmap Index Scan on checkin_date_idx (cost=0.00..10336.14 rows=492770 width=0) (actual time=26.349..26.350 rows...
19	Index Cond: ((checkin > '2021-05-01'::date) AND (checkin < '2021-10-01'::date))
20	-> Hash (cost=14.20..14.20 rows=137 width=16) (actual time=0.119..0.120 rows=137 loops=1)
21	Buckets: 1024 Batches: 1 Memory Usage: 15kB
22	-> Index Scan using roomtype_roomid on room r (cost=0.14..14.20 rows=137 width=16) (actual time=0.014..0.098 rows=13...
23	-> Index Scan using id_hotelbooking_idx on hotelbooking hb (cost=0.43..1.27 rows=1 width=8) (actual time=0.009..0.009 rows=0 l...
24	Index Cond: (idhotelbooking = rb."hotelbookingID")
25	Filter: payed
26	Rows Removed by Filter: 1
27	Planning Time: 0.910 ms
28	Execution Time: 456.461 ms

Συμπεράσματα

Συμπεραίνουμε πως περισσότερα ευρετήρια σε διαφορετικούς πίνακες δε σημαίνει πως θα βοηθήσουν την απόδοση του ερωτήματος μας αλλά αντιθέτως μπορεί να την κάνουν χειρότερη. Καταλήγουμε, λοιπόν, στο ότι διαφορετικά ευρετήρια είναι καλά για διαφορετικές περιπτώσεις ερωτημάτων και υποερωτημάτων.

Από τα πειράματά μας σε αυτό το μέρος της εργαστηριακής άσκησης, πιστεύουμε πως τα *B-+Tree* ευρετήρια είναι ίσως τα ιδανικότερα για τους περισσότερους τύπους ερωτημάτων και τα αποτελέσματα που παρουσιάσαμε παραπάνω επιβεβαιώνουν αυτή την τοποθέτηση. Με την ομαδοποίηση **(Clustering)** τα ερωτήματα γίνονται ακόμα πιο γρήγορα από ότι με απλά *Indexes* και στο συγκεκριμένο ερώτημα παρατηρούμε μείωση της ολοκλήρωσης του ερωτήματος κατά 20%-30% της πιο αποδοτικής περίπτωσης .

ΕΥΧΑΡΙΣΤΟΥΜΕ ΓΙΑ ΤΟΝ ΧΡΟΝΟ ΣΑΣ!