ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΜΜΥ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ
«ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ» – ΠΛΗ303
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2020-2021

ΔΙΔΑΣΚΩΝ: Δεληγιαννάκης Αντώνιος
ΕΡΓΑΣΤΗΡΙΑΚΟ ΠΡΟΣΩΠΙΚΟ: Καζάσης Φώτιος, Παππάς Νικόλαος

Ομάδα Χρηστών 126
Πέτρου Δημήτριος - 2018030070
Τσιπουράκη Αλεξάνδρα - 2018030089

Α΄ Φάση

Η υλοποίηση των ερωτημάτων της Α΄ φάσης πραγματοποιήθηκε με κατάτμηση του εκάστοτε προβλήματος σε υποσυναρτήσεις που τελικά συνιστούσαν μια ολοκληρωμένη συνάρτηση, όπου αυτό ήταν απαραίτητο και βοηθητικό. Παρακάτω παρατίθεται η αντιστοιχία συναρτήσεων και ερωτημάτων καθώς και ολοκληρωμένη λίστα των συναρτήσεων που χρησιμοποιεί κάθε ερώτημα.

Ερώτημα 2

2.1	`edit_client_2_1 (args…)`
	`├── insert_client_2_1 (args…)`
	`├── update_client_2_1 (args…)`
	`└── delete_client_2_1 (args…)`

2.2	`generate_bookings_2_2 (idhotel integer,sdate date,edate date,numofbooks integer)`
	`├── gethotelemployees (idhotel integer)`
	`├── selectrandomhotelemployee (idhotel integer)`
	`├── find_roombookings_of_hotel (idhotel integer)`
	`└── find_rate_of_room (idhotel integer, idroom integer)`

Ερώτημα 3

3.1	`find_discounts_3_1 ()`
3.2	`find_hotel_amenities_3_2 (hname varchar,hasstars varchar)`
3.3	`find_max_discount_rooms_of_hotels ()`

3.4	`find_hotel_bookings_3_4 (hotelIDarg integer)`
	`├── checkifclient (idclient integer)`
	`└── get_name_by_id (id integer)`

3.5	`find_hotel_empty_activities_3_5 (idhotel integer)`
3.6	`find_subtypes_offacilities_3_6 (nameOfFacility varchar)`

3.7	find_hotels_with_facilities_3_7 (roomheadfacil varchar, hotelheadfacil varchar)
3.8	find_hotel_available_type_3_8()

Ερώτημα 4

4.1	calculate_activities_ofclient_4_1 (hotelid integer) └── gethotelclients (idhotel integer)
4.2	calculate_avg_age_4_2 (roomtypearg varchar)
4.3	calculate_cheapest_rate_for_country_4_3 (countryname varchar)
4.4	calculate_profitable_hotels_percity_4_4 ()
4.5	calculate_completeness_4_5 (hotelid integer, yearArg integer) └── find_roombookings_of_hotel (idhotel integer)

Ερώτημα 5

5.1	TRIGGER FUNCTION: record_transaction_5_1 () TRIGGER: record_transaction_of_booking_5_1()
5.2	TRIGGER FUNCTION 1: check_validity_of_canceldate_change_5_2 () TRIGGER 1: validate_booking_date_changes_5_2 () ON hotelbooking TRIGGER FUNCTION 2: check_validity_of_roombooking_changes_5_2() TRIGGER 2: validate_booking_date_changes_5_2 () ON roombooking
5.3	TRIGGER FUNCTION 1: update_totalamount_for_booking_5_3 () TRIGGER 1: totalamount_updater_5_3 () TRIGGER FUNCTION 2: update_payment_changes_5_3 () └── find_cancel_date_of_booking (hbookingid integer) TRIGGER 2: updater_payment_changes_5_3() TRIGGER FUNCTION 3: overlap_check_5_3 () TRIGGER 3: overlap_checker_5_3 ()

Ερώτημα 6

6.1

```
VIEW TABLE NAME:
   view_available_rooms_ofhotels_6_1
```

6.2

```
VIEW TABLE NAME:
   view_hotel_weektable_6_2
   └── show_week_table_6_2(idhotel integer)
       └── get_week_bookings_6_2(sdate date, edate date, idhtl integer)
           └── find_detailed_bookings_of_hotel(idhotel integer)

UPDATE TRIGGER FUNCTION:
   save_weektable_changes_6_2()
UPDATE TRIGGER:
   weektable_updater_6_2()
```

Β΄ Φάση

1. Διασύνδεση Java εφαρμογής με DBMS PostgreSQL

Η υλοποίηση που συντάχθηκε σε γλώσσα Java είναι μια σχετικά απλή εφαρμογή κονσόλας που προσφέρει μια βασική διεπαφή με τον χρήστη . Σκοπός της, είναι η διασύνδεση και εκτέλεση SQL queries σε ένα σύστημα σχεσιακής βάσης δεδομένων PostgreSQL.

Η εφαρμογή αρχικά εμφανίζει ένα σύντομο μενού χρήστη το οποίο παρέχει επιλογές για εισαγωγή στοιχείων μιας βάσης προς σύνδεση και για εκτέλεση ερωτημάτων πάνω σε αυτήν. Το ίδιο μενού, επανεμφανίζεται στην κονσόλα όταν ολοκληρωθεί κάποια επιλεγμένη λειτουργία ή όταν ο χρήστης δώσει είσοδο ΄0΄ ενώ του ζητείται σε κάποιο σημείο εισαγωγή δεδομένων από το πληκτρολόγιο (διακοπή επιλεγμένης λειτουργίας).

Για την εκτέλεση των queries που υλοποιήθηκαν, ο χρήστης αρχικά θα πρέπει να δώσει ένα πρόθεμα ονόματος ξενοδοχείου προκειμένου να επιλέξει ένα εξ αυτών, το οποίο θα επιστραφεί από ένα αντίστοιχο query. Αφού επιλεγεί ένα ξενοδοχείο, στην κονσόλα εμφανίζεται κατάλληλο μενού, μέσα από το οποίο διαχειρίζονται τα queries του προαναφερθέντος ξενοδοχείου. Η επιλογή εκτέλεσης κάποιου query εμφανίζει τα σχετικά αποτελέσματα (αν είναι αναγκαίο) και ζητάει από το χρήστη την απαραίτητη πληροφορία για να ολοκληρωθεί το ερώτημα.

Η επικοινωνία της Java με τον PostgreSQL Server επιλέχθηκε να γίνει μέσω κατασκευασμένων συναρτήσεων Pl/pgSQL. Προκειμένου να αποδεσμευτεί η εφαρμογή από την διαδικασία σύνθεσης πολύπλοκων queries και την εκτέλεσή τους στον server, υλοποιήθηκαν εντός της βάσης συναρτήσεις ,οι οποίες με την σειρά τους καλύπτουν την ζητούμενη λειτουργικότητα σε επίπεδο ανάκτησης ή αποθήκευσης δεδομένων από και προς το database.

Κατα την υλοποίηση του κώδικα εντοπίστηκε μια ιδιαιτερότητα στη διασύνδεση μεταξύ της Java και της Pl/pgSQL. Όπως είναι γνωστό, η Pl/pgSQL δεν μπορεί να αναγνωρίσει ονόματα columns με συνύπαρξη πεζών και κεφαλαίων αν αυτά δεν περικλείονται από διπλά εισαγωγικά (πχ “idHotel”). Τα διπλά εισαγωγικά αποτελούν ειδικό χαρακτήρα της Java για τα Strings. Θα μπορούσε κανείς εντός ενός String να χρησιμοποιήσει τον ειδικό delimiter ΄\΄ για να συμπεριλάβει σε αυτό διπλά εισαγωγικά ή άλλον ειδικό χαρακτήρα. Κάτι τέτοιο ωστόσο, κατά την αποστολή του query σε String μορφή προς τη βάση οδηγεί στην παράλειψη των ειδικών χαρακτήρων, γεγονός, που δεν επιτρέπει στην SQL να αναγνωρίζει τελικά τα ονόματα ορισμένων στηλών. Μια ενδεχόμενη λύση θα ήταν να γίνει ανακατασκευή όλων των πινάκων όπου συμπεριλαμβάνονται στήλες με ονόματα τέτοιας μορφής, δηλαδή upper case and lower case γράμματα , κάτι τέτοιο όμως σε τόσο προχωρημένο στάδιο σχεδίασης της βάσης, θα ήταν κάθε άλλο παρά λύση. Επιλέχθηκε επομένως να δημιουργηθούν συναρτήσεις που να επιτρέπουν την διεπαφή της Java με την SQL χωρίς την μεσολάβηση ονομάτων arguments ή return types με κεφαλαία γράμματα.

2. Φυσικός σχεδιασμός βάσης - Μελέτη απόδοσης ερωτήσεων

Σκοπός της ζητούμενης μελέτης είναι η ανάλυση της εκτέλεσης ενός σύνθετου ερωτήματος SQL πάνω σε μια βάση δεδομένων που να επεξεργάζεται αρκετά εκατομμύρια tuples για παραγωγή αποτελέσματος. Οι βασικοί πίνακες που συμμετέχει στο ερώτημα είναι ο **roombooking** και ο **hotelbooking**. Το ερώτημα αφορά την ταξινόμηση των εσόδων που προκύπτουν από τις κρατήσεις ενός συγκεκριμένου τύπου δωματίου ανά μήνα μέσα σε ένα συγκεκριμένο χρονικό παράθυρο. Σε SQL μεταφράζεται ως εξής:

```
SELECT (SELECT EXTRACT(MONTH FROM "checkin")) as month,
       SUM(totalamount)as Income
FROM roombooking
       INNER JOIN room
       ON room."idRoom" = roombooking."roomID"
       INNER JOIN hotelbooking
       ON hotelbooking.idhotelbooking = roombooking."hotelbookingID"
WHERE (checkin,checkout) OVERLAPS ('2021-06-01'::date,'2021-06-03'::date) AND roomtype ='Twin'
GROUP BY (SELECT EXTRACT(MONTH FROM "checkin"))
```

α) EXPLAIN ANALYZE χωρίς τεχνικές βελτιστοποίησης

Αρχικά με την χρήση της EXPLAIN ANALYZE εξετάστηκαν ορισμένες παράμετροι εκτέλεσης του παραπάνω query χωρίς καμία τεχνική βελτιστοποίησης. Παράχθηκαν τα παρακάτω αποτελέσματα:

	QUERY PLAN text
1	HashAggregate (cost=816370.91..816486.81 rows=4636 width=12) (actual time=5824.456..5850.098 rows=12 loops=1)
2	Group Key: (SubPlan 1)
3	-> Gather (cost=1041.35..812798.75 rows=714431 width=12) (actual time=1.337..5805.130 rows=92633 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Nested Loop (cost=41.35..729639.19 rows=297680 width=8) (actual time=440.883..5699.846 rows=30878 loops=3)
7	-> Hash Join (cost=40.91..585002.13 rows=297680 width=8) (actual time=440.128..5278.251 rows=30878 loops=3)
8	Hash Cond: (roombooking."roomID" = room."idRoom")
9	-> Parallel Seq Scan on roombooking (cost=0.00..571695.25 rows=5043188 width=12) (actual time=369.519..5196.251 rows=523251 loops=3)
10	Filter: "overlaps"((checkin)::timestamp with time zone, (checkout)::timestamp with time zone, ('2021-06-01'::date)::timestamp with time zone, ('2021-06-03'::date)::timestamp w...
11	Rows Removed by Filter: 5537634
12	-> Hash (cost=39.41..39.41 rows=120 width=4) (actual time=0.536..0.537 rows=120 loops=3)
13	Buckets: 1024 Batches: 1 Memory Usage: 13kB
14	-> Seq Scan on room (cost=0.00..39.41 rows=120 width=4) (actual time=0.131..0.500 rows=120 loops=3)
15	Filter: ((roomtype)::text = 'Twin'::text)
16	Rows Removed by Filter: 1913
17	-> Index Scan using "idhotelbooking_PK" on hotelbooking (cost=0.43..0.49 rows=1 width=8) (actual time=0.013..0.013 rows=1 loops=92633)
18	Index Cond: (idhotelbooking = roombooking."hotelbookingID")
19	SubPlan 1
20	-> Result (cost=0.00..0.01 rows=1 width=8) (actual time=0.000..0.000 rows=1 loops=92633)
21	Planning Time: 0.603 ms
22	Execution Time: 5850.282 ms

Παρατηρήθηκαν τα εξής:

- Το δέντρο που αναπαριστά τις ενέργειες που υλοποιούν το παραπάνω query έχει ως εξής:

```
HashAggregate
└── Gather
     └── Nested Loop
          ├── Index Scan
          └── Hash Join
               ├── Parallel Seq Hash
               └── Hash
                    └── Seq Scan
```

  Η σειρά εκτέλεσης των κόμβων είναι inside-out.

- Ο χρόνος εκτέλεσης του query κυμάνθηκε στα **5.8s** όπως προέκυψε μετά από 6 εκτελέσεις της EXPLAIN ANALYZE

- Τα Nodes που εκτελέστηκαν είχαν την εξής σειρά:

  1. Αρχικά με τη χρήση Sequential Scan στον πίνακα room, τα tuples περνάνε από φίλτρο ώστε να διατηρηθούν μόνο αυτά που ικανοποιούν την συνθήκη **roomtype=΄Twin΄**. Η διαδικασία αυτή διαρκεί συνολικά **0.626ms** και τελικά επιλέγονται από τον πίνακα **120 γραμμές** ύστερα από **3 loops**.

  2. Με ένα Parallel Seq Scan στον πίνακα roombooking, τα tuples που δεν ικανοποιούν την συνθήκη για την επικάλυψη ημερομηνιών αφαιρούνται (αρ.tuples=5537778). Η διαδικασία αυτή διαρκεί συνολικά **5.200ms** και τελικά επιλέγονται από τον πίνακα **523251** γραμμές ύστερα από **3 loops**.

  3. Η διαδικασία του Hash Join για τους πίνακες room και roombooking διαρκεί συνολικά **5280ms** (αθρ = SeqScan+Parallel SeqScan) και παράγει τελικά **30878 tuples** μετά απο **3 loops**. Η διαδικασία του Hash Join περιλαμβάνει και την Index Scan για την σύνδεση των δύο παραπάνω πινάκων χρονικό κόστος περίπου **0.01ms** εξαιτίας του ευρετηρίου του foreign key idhotelbooking.

  4. Η Nested Loop ολοκληρώνεται τελικά σε περίπου 5700ms, χρόνος που αντιστοιχεί περίπου στο 97% του συνολικού χρόνου εκτέλεσης του query.

  5. Τέλος η HashAggregate φέρνει εις πέρας το query ολοκληρώνοντας το με την συνθήκη του GROUP BY σε μόλις περίπου **45ms**.

- Λαμβάνοντας υπόψιν όλα τα παραπάνω παρατηρούμε ότι το 90% του χρόνου εκτέλεσης του query καταναλώνεται εντός του Nested Loop και κυρίως με την Parallel Seq Scan. Η ευρετηριοποίηση του πίνακα roombooking με βάση τις ημερομηνίες checkin checkout χρησιμοποιώντας μια μέθοδο indexing κατάλληλη για αναζήτηση σε εύρος θα μπορούσε να συνεισφέρει στη μείωση αυτού του χρόνου.

β) EXPLAIN ANALYZE χωρις τεχνικές βελτιστοποίησης & parallel workers

Με τη χρήση της εντολής `SET max_parallel_workers_per_gather = 0` απενεργοποιήθηκε η δυνατότητα δημιουργίας παράλληλων πλάνων εκτέλεσης πριν την εκτέλεση της EXPLAIN ANALYZE. Η ανάλυση προέκυψε ως εξής:

	QUERY PLAN text
1	HashAggregate (cost=1405434.72..1405550.62 rows=4636 width=12) (actual time=16401.506..16401.532 rows=12 loops=1)
2	Group Key: (SubPlan 1)
3	-> Hash Join (cost=1091877.22..1401862.56 rows=714431 width=12) (actual time=10865.222..16373.199 rows=92633 loops=1)
4	Hash Cond: (hotelbooking.idhotelbooking = roombooking."hotelbookingID")
5	-> Seq Scan on hotelbooking (cost=0.00..173633.41 rows=10006441 width=8) (actual time=0.018..1956.762 rows=10006212 loops=1)
6	-> Hash (cost=1080155.84..1080155.84 rows=714431 width=8) (actual time=10810.706..10810.707 rows=92633 loops=1)
7	Buckets: 131072 Batches: 16 Memory Usage: 1251kB
8	-> Hash Join (cost=40.91..1080155.84 rows=714431 width=8) (actual time=0.872..10660.285 rows=92633 loops=1)
9	Hash Cond: (roombooking."roomID" = room."idRoom")
10	-> Seq Scan on roombooking (cost=0.00..1048276.60 rows=12103652 width=12) (actual time=0.094..10515.840 rows=1569752 loops=1)
11	Filter: "overlaps"((checkin)::timestamp with time zone, (checkout)::timestamp with time zone, ('2021-06-01'::date)::timestamp with time zone, ('2021-06-03'::date)::timestamp w...
12	Rows Removed by Filter: 16612903
13	-> Hash (cost=39.41..39.41 rows=120 width=4) (actual time=0.502..0.502 rows=120 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 13kB
15	-> Seq Scan on room (cost=0.00..39.41 rows=120 width=4) (actual time=0.034..0.470 rows=120 loops=1)
16	Filter: ((roomtype)::text = 'Twin'::text)
17	Rows Removed by Filter: 1913
18	SubPlan 1
19	-> Result (cost=0.00..0.01 rows=1 width=8) (actual time=0.000..0.000 rows=1 loops=92633)
20	Planning Time: 0.424 ms
21	Execution Time: 16401.846 ms

Όπως παρατηρείται, ο χρόνος εκτέλεσης τριπλασιάστηκε ενώ το δέντρο ενεργείων που συνιστούν το query άλλαξε αφού πλέον δεν δύναται να επιλεχθούν αλγόριθμοι παράλληλης αναζήτησης. Οι αριθμοί των συλλεγμένων tuples ανά Node προφανώς δεν αλλάζουν εφόσον το query απλά υλοποιείται με διαφορετικό τρόπο και όχι διαφορετικά κριτήρια. Το κόστος όλου του query έχει σχεδόν διπλασιαστεί φτάνοντας τιμή 1405434 έναντι του 816486 που ήταν στην περίπτωση πολλαπλών workers. Στην συνέχεια πραγματοποιήθηκε δοκιμή με αριθμό parallel worker ίσο με 1024 (max τιμή). Το query χρησιμοποίησε 5 workers που δεν μείωσαν σημαντικά το κόστος του και τον χρόνο εκτέλεσης του. Συμπερασματικά ένας αριθμός workers περί τον αριθμό 2 είναι επαρκής για την εκτέλεση του συγκεκριμένου ερωτήματος προς τη βάση.

γ) EXPLAIN ANALYZE με τεχνικές βελτιστοποίησης βασισμένες σε INDEXES

Με στόχο τη μείωση του χρόνου εκτέλεσης και την ελαχιστοποίηση του κόστους του ερωτήματος, όπως σχολιάστηκε παραπάνω, δημιουργήθηκαν αρχικά τα παρακάτω ευρετήρια προκειμένου να διευκολυνθεί η εκτέλεση των δύο JOIN. Τα indexes εφόσον θα χρησιμοποιηθούν σε συγκριτές ισότητας είναι INDEXES απλού τύπου.

```
CREATE INDEX roomid_idx ON roombooking("roomID");
CREATE INDEX idroom_idx ON room("idRoom");
CREATE INDEX idhbook_idx ON hotelbooking("idhotelbooking");
CREATE INDEX hbookid_idx ON roombooking("hotelbookingID");
```

Επιλέχθηκε επίσης να γίνει η δημιουργία ενός ευρετηρίου για τις ημερομηνίες με την χρήση B-Tree καθώς ο τελεστής overlaps συγκρίνει εύρος και ανισότητες, γεγονός που καθιστά ακατάλληλη, μια άλλη επιλογή δομής όπως το Hash.

```
CREATE INDEX dates_idx ON roombooking USING btree(checkin,checkout);
```

Με την εκτέλεση της EXPLAIN ANALYZE προέκυψαν τα εξής:

	QUERY PLAN text
1	HashAggregate (cost=296027.41..296143.31 rows=4636 width=12) (actual time=2154.371..2154.392 rows=12 loops=1)
2	Group Key: (SubPlan 1)
3	-> Nested Loop (cost=0.87..294238.66 rows=357750 width=12) (actual time=0.703..2116.758 rows=92633 loops=1)
4	-> Nested Loop (cost=0.44..103142.13 rows=357750 width=8) (actual time=0.603..1441.965 rows=92633 loops=1)
5	-> Seq Scan on room (cost=0.00..39.41 rows=120 width=4) (actual time=0.021..0.671 rows=120 loops=1)
6	Filter: ((roomtype)::text = 'Twin'::text)
7	Rows Removed by Filter: 1913
8	-> Index Scan using roomid_idx on roombooking (cost=0.44..829.38 rows=2981 width=12) (actual time=0.199..11.908 rows=772 loops=120)
9	Index Cond: ("roomID" = room."idRoom")
10	Filter: "overlaps"((checkin)::timestamp with time zone, (checkout)::timestamp with time zone, ('2021-06-01'::date)::timestamp with time zone, ('2021-06-03'::date)::timestamp with ...
11	Rows Removed by Filter: 8179
12	-> Index Scan using idhbook_idx on hotelbooking (cost=0.43..0.52 rows=1 width=8) (actual time=0.006..0.006 rows=1 loops=92633)
13	Index Cond: (idhotelbooking = roombooking."hotelbookingID")
14	SubPlan 1
15	-> Result (cost=0.00..0.01 rows=1 width=8) (actual time=0.000..0.000 rows=1 loops=92633)
16	Planning Time: 0.960 ms
17	Execution Time: 2154.577 ms

Όπως φαίνεται ο χρόνος εκτέλεσης του query έχει μειωθεί παραπάνω από το μισό, όντας ίσος με 2.1s. Πλέον ο βελτιστοποιητής δεν επιλέγει την εκτέλεση Parallel Seq Scan παρά ακολουθεί ένα απλούστερο Index Scan σύμφωνα με το ευρετήριο ημερομηνιών. Σημαντική δεν είναι μόνο η διαφορά στο χρόνο εκτέλεσης αλλά και στο συνολικό κόστος του ερωτήματος, το οποίο ανέρχεται στα 296027 έναντι του προηγούμενου 816486.

δ) EXPLAIN ANALYZE με τεχνικές βελτιστοποίησης με χρήση INDEXES και CLUSTERING

Προς περαιτέρω βελτίωση της απόδοσης του ερωτήματος μέσω της τεχνικής συσταδοποίησης μορφοποιήθηκε ο τρόπος αποθήκευσης του πίνακα roombooking σύμφωνα με το ευρετήριο που κατασκευάστηκε για τις ημερομηνίες κρατήσεων. Με την τεχνική αυτή, ημερομηνίες που είναι κοντά μεταξύ τους θα συσταδοποιηθούν μαζί, με αποτέλεσμα το query εκτελώντας έλεγχο για overlap το query να μην ανατρέχει σε πολλά σημεία του πίνακα.

```
CLUSTER roombooking USING dates_idx;
```

Η EXPLAIN ANALYZE έδωσε τα παρακάτω αποτελέσματα:

	QUERY PLAN text
1	HashAggregate (cost=268218.08..268333.98 rows=4636 width=12) (actual time=9970.643..9970.671 rows=12 loops=1)
2	Group Key: (SubPlan 1)
3	-> Nested Loop (cost=0.87..266429.33 rows=357750 width=12) (actual time=0.863..9888.466 rows=92633 loops=1)
4	-> Nested Loop (cost=0.44..75332.79 rows=357750 width=8) (actual time=0.365..8537.952 rows=92633 loops=1)
5	-> Seq Scan on room (cost=0.00..39.41 rows=120 width=4) (actual time=0.187..0.533 rows=120 loops=1)
6	Filter: ((roomtype)::text = 'Twin'::text)
7	Rows Removed by Filter: 1913
8	-> Index Scan using roomid_idx on roombooking (cost=0.44..597.63 rows=2981 width=12) (actual time=0.720..70.983 rows=772 loops=120)
9	Index Cond: ("roomID" = room."idRoom")
10	Filter: "overlaps"((checkin)::timestamp with time zone, (checkout)::timestamp with time zone, ('2021-06-01'::date)::timestamp with time zone, ('2021-06-03'::date)::timestamp with ...
11	Rows Removed by Filter: 8179
12	-> Index Scan using idhbook_idx on hotelbooking (cost=0.43..0.52 rows=1 width=8) (actual time=0.012..0.012 rows=1 loops=92633)
13	Index Cond: (idhotelbooking = roombooking."hotelbookingID")
14	SubPlan 1
15	-> Result (cost=0.00..0.01 rows=1 width=8) (actual time=0.000..0.001 rows=1 loops=92633)
16	Planning Time: 0.769 ms
17	Execution Time: 9970.859 ms

Τα αποτελέσματα δεν ήταν τα αναμενόμενα, ο χρόνος εκτέλεσης αυξήθηκε πέρα από την αρχική του τιμή δίχως βελτιστοποίηση, ενώ η μείωση του κόστους του query δεν υπήρξε εξαιρετικά σημαντική. Κρίθηκε σκόπιμο να επιλεχθεί μια διαφορετική τεχνική συσταδοποίησης. Η συσταδοποιήση σύμφωνα με το idhotelbooking στον πίνακα roombooking επαναφέρει τον χρόνο εκτέλεσης στο έως τώρα βέλτιστο επίπεδο.

```
CLUSTER roombooking USING hbookid_idx;
```

Τέλος πραγματοποιήθηκε συσταδοποίηση με βάση ένα νέο ευρετήριο που ήταν βασισμένο στο έτος της κράτησης προκειμένου να απορριφθούν από το φίλτρο κρατήσεις που δεν ανήκουν στο έτος αναζήτησης. Κάτι τέτοιο ωστόσο, δεν έδωσε καλύτερα αποτελέσματα από τα ήδη βέλτιστα.

Συμπερασματικά καλύτερη ως προς το κόστος αλλά και ως προς το χρόνο εκτέλεσης παραμένει η συσταδοποίηση με βάση το ID της κράτησης. Αυτό ενδεχόμενως οφείλεται στην ένωση του πίνακα roombooking με τον hotelbooking ο οποίος ακολουθεί την ίδια λογική clustering.

ε) EXPLAIN ANALYZE με επιλεκτική απενεργοποίηση αλγορίθμων

Προκειμένου να μελετηθεί σε μεγαλύτερο βάθος η συμπεριφορά του optimizer απενεργοποιήθηκαν κατ΄επιλογή οι αλγόριθμοι που έχει στη διάθεση του για την κατασκευή ενός query.

- **Nest Loop**

  Απενεργοποιώντας τη δυνατότητα συμμετοχής του αλγορίθμου Nest Loop στην υλοποίηση του query, ο χρόνος εκτέλεσης πενταπλασιάζεται σε σχέση με τη βελτιστοποιημένη εκδοχή αφού ανέρχεται στα 9.6s. Ο optimizer ωστόσο δεν αποτυγχάνει στο να φέρει το ερώτημα εις πέρας χρησιμοποιώντας στη θέση των nested loops ένα Parallel Hash Join και ένα Parallel Hash.

- **Nest Loop & Hash**

  Με αφορμή το προηγούμενο αποτελέσμα επιλέγεται να απενεργοποιηθεί παράλληλα με το Nest Loop και το Hash Join για να μελετηθεί η συμπεριφορά του optimizer και υπό αυτές τις συνθήκες. Τα αποτελέσματα αποδείχθηκαν σχετικά καλύτερα από την προηγούμενη δοκιμή χωρίς ωστόσο σημαντικές διαφορές ως προς το χρόνο εκτέλεσης. Το κόστος αυτή τη φορά ήταν εξαπλάσιο του βέλτιστου, αφού ανήλθε στο 1101924. Επιπροσθέτως κάτι το οποίο παρατηρήθηκε ήταν οτι προέκυψε χρήση του δίσκου σε ένα Sort Node που ανήλθε στα 1640kB.

- **Sort**

  Με επαναφορά των Nest Loop και Parallel Hash στη διάθεση του optimizer απενεργοποιήθηκε η Sort. Η επίπτωση στην απόδοση δεν ήταν μεγάλη. Αντίθετα με την προηγούμενη περίπτωση, όλες οι παράμετροι αξιολόγησης της εκτέλεσης βελτιώθηκαν. Ο optimizer στη θέση της Sort χρησιμοποίησε Seq Scan.

ΠΑΡΑΡΤΗΜΑ: Βοηθητικές Συναρτήσεις

add_activity(args) : Εισαγωγή νέου activity με βάση τα arguments
checkifclient(args) : Έλεγχος αν ένα id ανήκει σε πελάτη του ξενοδοχείου
convertinttoday(args) : Μετατροπή ακεραίου σε ημέρα της εβδομάδας (πχ 1->Δευτέρα)
find_booking_whomanages(args) : Έυρεση του εργαζομένου που διαχειρίζεται μια κράτηση
find_cancel_date_of_booking(args) : Εύρεση της ημερομηνίας ακύρωσης μιας κράτησης
find_hotel_manager(args) : Εύρεση του διευθυντή ενός ξενοδοχείου
find_hotel_ofbookings(args) : Εύρεση του ξενοδοχείο στο οποίο ανήκει μια κράτηση
find_roombookings_of_hotel(args) : Εύρεση των κρατήσεων των δωματίων ενός ξενοδοχείου
find_roombooking_hotel_period(args) : Εύρεση των κρατήσεων ενός ξενοδοχείου σε ένα συγκερκιμένο διάστημα
find_bookings_of_client(args) : Εύρεση των κρατήσεων ενός συγκεκριμένου πελάτη
find_available_rooms_of_hotel(args) : Εύρεση των διαθεσιμων δωματιών ενός ξενοδοχείου για ενα συγκεκριμενο χρονικο διαστημα
find_hotel_clients_prefix(args) : Εύρεση πελατών ενός ξενοδοχείου σύμφωνα με πρόθεμα επωνύμου
find_next_week() : Εύρεση της πρώτης και της τελευταίας μέρας της επόμενης βδομάδας από την τρέχουσα
generate_manages() : Ανάθεση των κρατήσεων κάθε ξενοδοχείου σε τυχαίο εργαζόμενο του
get_hotel_rooms(args) : Εύρεση των ids των δωματίων ενός ξενοδοχείου
get_name_by_id(args) : Εύρεση του ονόματος ενός person με βάση το id του.
gethotelclients(args) : Εύρεση των πελατών ενός ξενοδοχείο
gethotelemployees(args) : Εύρεση των εργαζομένων ενός ξενοδοχείου
selectrandomhotelclient(args) : Επιλογή ενός τυχαίου πελάτη ενός ξενοδοχείου
selectrandomhotelemployees(args) : Επιλογή ενός τυχαίου εργαζομένου ενός ξενοδοχείου
show_allclients() : Εμφάνιση όλων των πελατών της βάσης
show_allemployees() : Εμφάνιση όλων των εργαζομένων της βάσης.
update_roombooking(args) : Ενημέρωση ενός roombooking