Αναφορά

Ονομα: Χριστοδουλίδης Κυριάκος **ΑΜ:** 2016030025
Ονομα: Γιώργος Δουκας **ΑΜ:** 2016030032
ΠΛΗ303 - Βάσεις Δεδομένων
Πολυτεχνειο Κρητης

May 23, 2021

Φάση Α

1.1

Για το πρώτο ερώτημα, δημιουργήθηκαν τα κατάλληλα tables, χωρίς δεδομένα, για τα κόκκινα τμήματα του ER σχήματος που μας δόθηκε, έχοντας τα σωστά relations.

2.1 (action_2_1)

Αυτό το ερώτημα, υλοποιήθηκε με 4 συναρτήσεις, μια Insert (insert_2_1), μια Update (update_2_1), μια Delete(delete_2_1) και μια που γίνεται η επιλογή αυτών των actions (action_2_1). Κατά την εισαγωγή, δημιουργείται ένα idPerson, το οποίο επειδή πρέπει να είναι unique το θέτουμε ως max+1 από τα ήδη υπάρχοντα idPerson, και εισάγονται όλα τα υπόλοιπα values κάνοντας κάποιους ελέγχους στις τιμές που πρέπει να είναι unique. Για το Update, πήραμε ως μη updatable το documentclient, ώστε να μπορουμε να βρούμε το person που γίνεται η αλλαγή. Στη συνέχεια ενημερώνουμε όλες τις τιμές τις οποίες δίνονται. Τέλος για το Delete διαγράφοντας το Person, επειδή το relation του Person-client και client-creditcard έχει cascade, διαγράφονται και τα δεδομενα που ζητούνται.

2.2 (n_hotel_rooms_insert_2_2)

Αρχικά, επιλέγεται μέσα στην χρονική περίοδο ένα τυχαίο διάστημα στο οποίο θα γίνονται οι κρατήσεις, και βρίσκονται όλα τα ελεύθερα δωμάτια για αυτό το διάστημα. Στη συνέχεια, επιλέγονται τυχαία 1-5 δωμάτια για την συμπλήρωση μιας κράτησης. Έπειτα ελέγχονται οι περιορισμοί πελάτη και υπεύθυνου ατόμου δωματιου που ζητούνται. Τελικά με τα values που έχουν δημιουργηθεί εφόσον ικανοποιηθούν οι περιορισμοί γίνεται η εισαγωγή μιας κράτησης (hotelbooking) και αντίστοιχα 1-5 roombooking, για αυτή την κράτηση.

3.1 (over_30_percent_discount_3_1)

Σε αυτό το ερώτημα με ένα join μεταξυ hotel και roomrate πήραμε τις χώρες/πόλεις με έκπτωση δωματίων πάνω από 30%.

3.2 (choose_and_facilities_3_2)

Σε αυτό το ερώτημα, ο χρήστης δίνει το πρόθεμα (prefix) του hotel name και τον αριθμό των αστερίων, και επιλέγονται απο το hotel τα αντίστοιχα tuples για αυτές του τις επιλογές. Έπειτα γίνεται join αυτό το αποτέλεσμα, με τα δωμάτια τύπου Studio κόστους μικρότερου των 80 ευρώ, που προσφέρουν πρωίνο και έχουν εστιατόριο.

3.3 (max_discount_per_room_3_3)

Σε αυτό το ερώτημα, βρήκαμε τις μέγιστες προσφορές δωματίων ανά τύπο, και τυπώνονται τα ξενοδοχεία που είχαν προσφόρα στον τύπο αυτό, ίσο με τον μέγιστο που βρήκαμε.

3.4 (hotel_reservation_3_4)

Σε αυτό το ερώτημα, αρχικά βρήκαμε ανά ξενοδοχείο όλες τις κρατήσεις του. Στη συνέχεια, από το bookedByClient που υπάρχει στο hotelbooking, βρήκαμε το clientFname και clientLname. Τέλος από το bookedByClient, ελέγχουμε με ένα case When αν είναι employee η client το id αυτό, και συμπληρώνουμε το BookedBy αναλόγως.

3.5 (activities_no_participants_3_5)

Σε αυτό το ερώτημα, αφού πρώτα κάναμε εισαγωγή activities για κάθε ξενοδοχείο στο table activity, ελέγχουμε ποιές από αυτές δεν έχουν κανέναν participant ακόμα.

3.6 (hotel_facility_types_3_6)

Σε αυτό το ερώτημα, έχοντας βρει τις διευκολύνσεις ξενοδοχείων πρώτης κατηγορίας, βρίσκουμε αναδρομικά, όλους τους υποτύπους διευκολύσεων αυτών.

3.7 (hotel_specifics_facilities_3_7)

Σε αυτό το ερώτημα, δίνοντας ώς είσοδο μια λίστα απο συγκεκριμένες διευκολύνσεις για room και για hotel, προβάλλονται τα ξενοδοχεία που παρέχουν όλες τις διευκολύνσεις αυτές.

3.8 (hotel_available_rtypes_3_8)

Σε αυτό το ερώτημα, εκμεταλευτήκαμε το ερώτημα 6.1, που επιστρέφει όλα τα ελεύθερα δωμάτια για όλα τα ξενοδοχεία αυτή την χρονική στιγμή. Έτσι ελέγχουμε τον αριθμό των ελεύθερων ανά τύπο δωματίων και το συγκρίνουμε με τον συνολικό αριθμό των τύπων δωματίων που έχει. Αν είναι ίσος εμφανίζεται στην αναζήτηση.

4.1 (participates_count_hotel_4_1)

Σε αυτό το ερώτημα, βρίσκαμε ανά person που σχετίζεται με κάποιο ξενοδοχείο πόσες φορές συμμετείχε σε δραστηριότητες αυτού. Αν δεν συμμετείχε σε καμία δραστηριότητα του ξενοδοχείου, τότε εμφανίζεται 0.

4.2 (find_average_ages_4_2)

Σε αυτό το ερώτημα, βρίσκουμε όλους τους πελάτες που έχουν κάνει κράτηση σε συγκεκριμένο τύπο δωματίου, και υπολογίζουμε τον μέσο όρο ηλικίας τους.

4.3 (find_min_cost_4_3)

Σε αυτό το ερώτημα, υπολογίζουμε την φθηνότερη τιμή δωματίου ανά τύπο στην χώρα που έδωσε ο χρήστης. Στην συνέχεια εμφανίζουμε όσους τύπους στην χώρα αύτη έχουν ίση τίμη με την φθηνότερη και την πόλη στην οποία βρίσκονται.

4.4 (hotel_above_average_4_4)

Σε αυτό το ερώτημα, υπολογίζουμε τις εισπράξεις του κάθε ξενοδοχείου με βάση το rate στο roombooking καθώς το totalamount ήταν null. Να σημειωθεί, ότι τα 100 ξενοδοχεία που μας δόθηκαν, ήταν όλα σε διαφορετική πόλη. Έτσι ο μέσος όρος σε κάθε πόλη ήταν τα έσοδα του ίδιου του ξενοδοχείου στην πόλη αυτή. Έφοσον, ζητούνται ξενοδοχεία με μεγαλύτερες εισπράξεις από τον μέσο όρο εισπράξεων ανά πόλη, τότε το αποτέλεσμα δεν επιστρέφει καμία γραμμή στο return table. Για τον έλεγχο του ερωτήματος, αλλάξαμε την τοποθεσία πόλης ενός ξενοδοχείου.

4.5 (percentage_of_full_rooms_4_5)

Σε αυτό το ερώτημα, για μια συγκεκριμένη χρονιά που δίνει ο χρήστης, βρίσκουμε ανά μήνα πόσα δωμάτια από κάθε ξενοδοχείο υπάρχει στο roombooking. Στη συνέχεια διαιρούμε αυτόν τον αριθμό με τον συνολικό αριθμό δωματιών του κάθε ξενοδοχείου, ώστε να υπολογίσουμε την πληρότητα.

5.1 (transaction_update_5_1)

Σε αυτό το ερώτημα, μόλις ο χρήστης κάνει κάποια πληρωμή, δηλαδή αλλάξει το payed από false σε true, και το paymethod στη μεθοδο πληρωμής, αυτόματα υπολογίζουμε το totalamount, αθροίζοντας τα rates για την κράτηση αυτή. Στη συνέχεια, καταγράφουμε στο Transaction table την συνδιαλλαγή που έγινε.

5.2 (rooms_xxxx_5_2)

Σε αυτό το ερώτημα, εφόσον συμπληρώσαμε το table manages, ελέγχαμε αν το δωμάτιο που θέλει να γίνει update διαχειρίζεται από κάποιο manager. Αν ναι τότε μπορούσε να γίνει οποιαδήποτε αλλαγή. Αν όχι, τότε το trigger άφηνε να γίνουν update μόνο ότι δεν παραβιάζει τους περιορισμούς που ζητήθηκαν.

5.3 (rate_trigger_5_3)

Σε αυτό το ερώτημα, έγινε ενα trigger στο roomrate. Αρχικά κατά το insert και το update γίνεται αυτόματα η ανανέωση του rate σύμφωνα με την τιμή που έχει το roomrate μαζί με την έκπτωση. Στην συνέχεια, για διαγραφές και ανανεώσεις σε πληρωμένες κρατήσεις, ανανεώνεται κατάληλλα ο πίνακας συνδιαλλαγών όταν χρειάζεται. Αν έχει περάσει το cancellationdate, τότε με το RAISE NOTICE εμφανίζεται το κατάληλλο μήνυμα. Τέλος, στην παράταση διαμονής, βρίσκουμε το αμέσως επόμενο checkin που έχει αν υπάρχει το συγκεκριμένο δωμάτιο, και αν αυτό είναι μικρότερο από το checkout που δώσει ο χρήστης, τότε, εμφανίζουμε πάλι με RAISE NOTICE ως διαθέσιμο μέχρι το checkin που βρήκαμε.

6.1 (available_rooms_6_1)

Σε αυτό το ερώτημα, δημιουργήσαμε μια όψη. Με την χρήση της συνάρτησης now() βρίσκουμε για κάθε ξενοδοχείο όλα τα διαθέσιμα δωμάτια μαζί με τον τύπο τους για αυτή την χρονική

στιγμή. Για να βρούμε μέχρι πότε είναι διαθέσιμο αυτό το δωμάτιο, βρίσκουμε όλα τα checkin που για κάθε δωμάτιο είναι μεγαλύτερα από το now(). Έπειτα γίνεται order by checkin, και distinct on(roomid ,checkin). Έτσι κρατάμε για κάθε δωμάτιο μόνο το αμέσως επόμενο checkin που υπάρχει. Αν δεν υπάρχει σημάινει ότι δεν θα είναι ποτε μη διαθέσιμο, όποτε τυπώνεται στο date 'infinite'.

6.2 (weekly_plan_6_2)

Σε αυτό το ερώτημα, υπολογιζουμε την πρώτη Κυριακή από το now(), και την χρησιμοποιούμε για να καθορίσουμε την εβδομάδα του εβδομαδαίου πλάνου. Στη συνέχεια για ένα τυχαίο ξενοδοχείο κάθε φορά, εμφανίζουμε για καθε μέρα της εβδομάδας, τα δωμάτια που έχουν κρατήσεις. Σε αυτά τα δωμάτια, για τις μέρες που είναι κρατημένα εμφανίζουμε το documentclient. Αν το δωμάτιο δεν είναι κρατημένο για όλη την βδομάδα, για τις μέρες που είναι ελεύθερο, εμφανίζεται 0 στο documentclient.

Insert Tuples in Tables with no Data (Red color tables)

Activity (activity_generator)

Για την εισαγωγη ενός activity, αρχικά, επιλέγουμε για μια μέρα μέσα στις επόμενες 30 μέρες, ενα χρονικό διάστημα από 8 το πρωί μέχρι 23 το βράδυ, μια κατηγορία από acitivity. Στην συνέχεια ελέγχουμε αν το random activity που δημιουργήθηκε καλύπτει τους περιορισμούς. Αν ναί τότε το εισάγουμε στο activity, αλλίως δημιουργέιται τυχαία άλλο και ξαναελέγχεται. Συνολικά δημιουργήσαμε 505 activities.

Participates (participates_xxxx_insert)

Για τους Participates, όσο αφορά τους responsibles, διαλέγουμε από τους employees του ξενοδοχείου τυχαία 1-10 εργαζόμενους και τους εισάγουμε στο acitivity αυτό. Για τους participants, παίρνουμε τυχαία από τους clients του ξενοδοχείου 0-50 άτομα, και εξίσου τους εισάγουμε στο participates.

Manages (insert_managers)

Για την συμπλήρωση του manages, επιλεξαμε τυχαία εργαζόμενοι του ξενοδοχείου, να αναλάμβανουν να κάνουν manage κάποια, 1 η περισσότερα hotelbookings.

Φάση Β

Explain Analyze

Query

```
EXPLAIN ANALYZE WITH roomtypes as(
        Select rate,
        Case When (checkin<'2021-07-22' and checkout>'2021-07-22')
Then
    TO_CHAR(checkin,'month'::text)
Else
    TO_CHAR(checkout,'month'::text)
End as month
        from room join roombooking on "idRoom"="roomID" where roomtype='Studio'
         and checkout>'2021-05-22' and checkin<'2021-07-22')

Select month,totalcost
    from
    (Select month, sum(rate) as totalcost from roomtypes   group by  month) as monthcost
     order by totalcost;
```

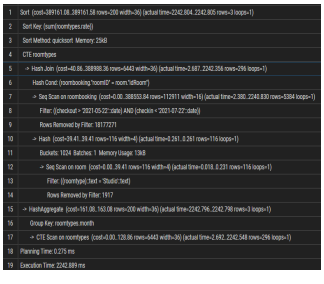
Figure 1: Question Query

Επιλογες του χρηστη:
checkin = '2021-05-22'
checkout = '2021-05-22'
roomtype = 'Studio'

Όσο αφορά το Query που ζητήθηκε να υλοποιήσουμε, χρειάστηκε να κάνουμε filtering σε ένα range query με βάση τα checkin και checkout από το table roombooking και ένα point query filtering στο roomtype, από το table room. Όσο αφορά τα join που έγιναν, ήταν ένα inner join μεταξυ του room και roombooking στο idRoom. Να σημειωθεί ότι για όλα τα ερωτήματα εκτελέστηκε ακριβώς το ίδιο query για τα παραπάνω inputs του χρήστη.

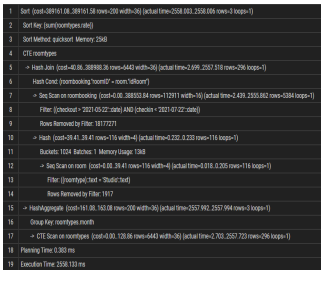
Default

Αρχικά απενεργοποιήσαμε τη δυνατότητα δημιουργίας παράλληλων πλάνων εκτέλεσης χρησιμοποιώντας την εντολή 'set max_parallel_workers_per_gather = 0' όπως αναφέρεται στην εκφώνηση. Έπειτα εκτελόντας τις εντολές EXPLAIN ANALYZE πήραμε τα παρακάτω αποτελέσματα.



(a)

(b)



(c)

Figure 2: Primary keys indexing

Παρόλο που στο roombooking υπάρχουν εκατομμύρια γραμμες με δεδομένα, οι παραπάνω χρόνοι σαφώς και μπορούν να βελτιωΘούν. Αυτό το συμπεραινουμε καθώς το indexing που γίνεται είναι πάνω στα primary keys, και στο Query το οποίο τρέχουμε βασίζεται στα checkin,checkout του roombooking και roomtype του room. Όποτε με αλλαγη των ευρετηρίων περιμένουμε να βελτιωθεί ο χρόνος εκτέλεσης του Query μας. Πιο συγκεκριμένα το actual time στο Seq scan στο roombooking είναι ουσιαστικά σχεδόν όλος ο χρόνος του execution time. Όποτε περιμένουμε ότι με κάποιο indexing στο checkin-checkout θα έχουμε τεράστια διαφορά στα αποτελέσματα. Μέσος χρόνος εκτέλεσης ειναι 2300 ms.

Indexing

Από τη θεωρία μας, την διάλεξη με τα Indexing, γνωρίζουμε 2 διαφορετικά είδη indexing. Το hash indexing και το btree indexing. Όταν στο Query κάνουμε filtering σε point queries τότε το hash έχει πολυπλοκότητα O(1) ενώ το Btree χρείαζεται χρόνο O(logN), οπότε σε αυτές τις περιπτωσεις προτιμούμε το Hash. Όταν στο Query κάνουμε filtering με range query τότε το Btree έχει πολυπλοκότητα πάλι O(logN) ενώ το Hash έχει πολυπλοκότητα O(N) καθώς κάνει σειριακή αναζήτηση. Άρα σε range query θα χρησιμοποιούμε μόνο το Btree και στα point queries μπορούμε να χρησιμοποιήσουμε και τα δύο αλλά είναι πιο αποδοτικό το hash.

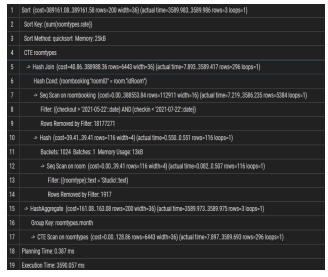
(a) (b) (c)

Figure 3: btree(checkout)

Αρχικά δοκιμάσαμε να κάνουμε btree στο checkout εφόσον το filtering ήταν range query. Όπως περιμέναμε λοιπόν, ο χρόνος βελτιώθηκε δραματικά, ωστόσο το αποτέλεσμα μπορεί να βελτιωθεί περαιτέρω. Να σημειωθεί ότι από τις διαλέξεις ξέρουμε ότι η πολυπλοκότητα στο query range με btree είναι O(logN). Ώστοσο αυτό δεν ισχυεί πάντα. Όσο πιο μεγάλο είναι το ευρος του range query, τόσο πιο μεγάλη η πολυπλοκότητα του. Αυτό είναι κάτι που το ελέγχει ο βελτιστοποιητής στο planning time. Όποτε, είδαμε δραματικές αλλαγές στον χρόνο επειδή το range query στην συγκριση που εγίνε με το checkout επιστράφηκε μικρό query. Μέσος χρόνος εκτέλεσης 20-30 ms

(a)

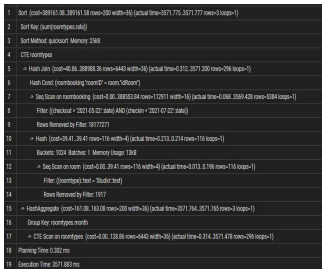
(b)



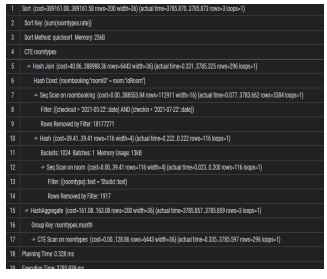
(c)

Figure 4: btree(checkin)

Σε αυτή την περίπτωση ο βελτιστοποιητής δεν επιλέγει το checkin indexing με btree που δημιουργήσαμε. Αυτό είναι κάτι που δεν περιμέναμε, αλλά εξηγειται η συμπεριφορα αυτη επειδή το εύρος τιμών που θα επιστραφεί μετα την συγκριση του checkin είναι τεράστια. Αυτό έχει ως αποτέλεσμα η πολυπλοκότητα εκτέλεσης με ευρετήριο το checkin να είναι τεράστια. Έτσι ο βελτιστοποιητής επιλέγει ως ευρετήριο το default που είναι το primary key του roombooking. Μέσος χρόνος εκτέλεσης 3200-3800 ms.



(a)



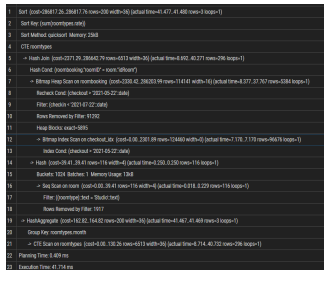
(b)

(c)

Figure 5: hash(checkout)

Όπως γνωρίζαμε και από την θεωρία το hash είναι πολυ κακό για filtering σε range query, κατί που αποδεικνυεται και πρακτικά. Εδω ο μέσος χρόνος εκτέλεσης ειναι 3700-4400 ms.

(a)



(b)

(c)

Figure 6: btree(checkin) and btree(checkout)

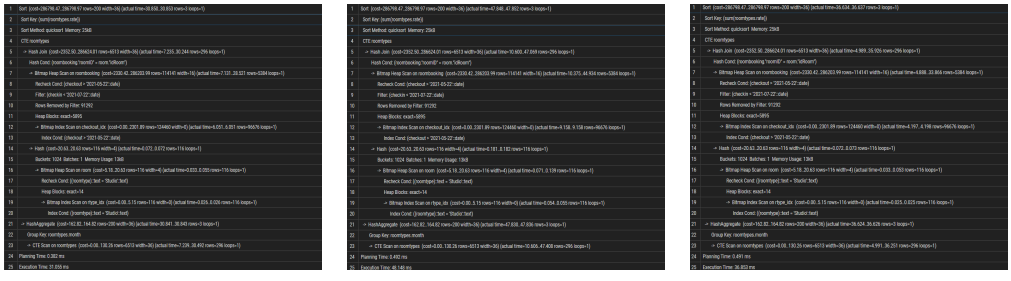
Όπως είπαμε και προηγουμενώς με τις επιλογές που έδωσε ο χρήστης ο βελτιστοποιητής θα επιλέξει το checkout indexing. Ώστοσο, με άλλες επιλογές του χρήστη θα μπορούσε να επιλεχτεί το checkin από τον βελτιστοποιήτη. Αν ήταν πιο αργά και τα δύο από το primary key indexing, τότε θα επέλεγε αυτό ο βελτιστοποιήτης. Για αυτό το λόγο, σε ερωτήματα με

πολλά δεδομένα είναι πολλές φορές καλό να προσθέτουμε indexing σε columns που χρησιμοποιούμε συχνά σε αναζητήσεις, και να αφήνουμε τον βελιστοποιήτη να επιλέγει κάθε φορά τον γρηγορότερο τρόπο εκτελέσης του query. Παρόλα αυτά το indexing αποθηκεύεται στον δίσκο, χώρος που μπορεί να μην είναι πάντα διαθέσιμος. Εφόσον λοιπον εμείς είχαμε πάντα το ίδιο Query προς μελετη, γνωρίζαμε ότι ο βελτιστοποιητής δεν θα το διαλεγέ πότε σαν λύση, όποτε το απορρίψαμε για την συνέχεια της μελέτης μας. Εδω ο μέσος χρόνος εκτέλεσης είναι 20-40 ms, παρόμοιος αυτού με μόνο το checkout σαν index.

(a) (b) (c)

Figure 7: btree(checkout) and hash(roomtype)

Εφόσον βρήκαμε το κατάλληλο indexing του roombooking, δοκιμάσαμε να κάνουμε hash στο roomtype που είναι στο table room. Δοκιμάσαμε πρώτα hash καθώς έχουμε filtering σε point query. Παρατηρούμε ότι ο βελτιστοποιήτης επιλέγει το roomtype indexing που δημιουργησαμε, παρόλα αυτά η διαφορά στον χρόνο είναι μικρότερη του 1 ms σε σχέση με προηγουμένως. Αυτό οφείλεται στο ότι το room έχει πολύ λίγα δεδομένα, όποτε η διαφορά είναι προφανώς λιγότερα αισθητή. Να σημειωθεί ότι επειδή το checkout και το roomtype είναι σε διαφορετικά table, για τον βελτιστοποιήτη είναι δύο ανεξάρτητες επιλογές μεταξύ τους. Μέσος χρόνος εκτέλεσης είναι 25 ms.



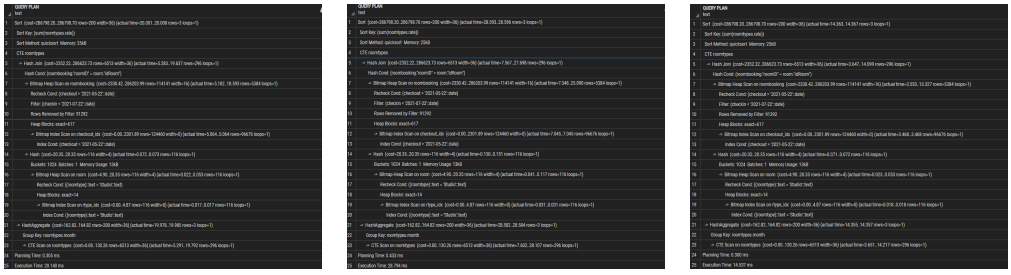
(a) (b) (c)

Figure 8: btree(roomtype) and btree(checkout)

Ελένξαμε λοιπόν btree indexing στο checkout, και επαληθεύσαμε οτι ξερουμε απο την θεωρία. Είναι μεν καλύτερο από το primary key indexing του room, ώστοσο, έχει χειρότερα αποτελέσματα, με μικρή διαφορά, από το hash indexing. Η διαφορά αυτή θα ήταν αισθητά μεγαλύτερη αν το room είχε περισσότερα δεδομένα. Όποτε καταλήγουμε ότι το hash indexing για το Query μας είναι καλύτερο από το btree indexing για το roomtype. Μέσος χρόνος εκτέλεσης ειναι 30-50 ms.

Clusters

Όσο αφορά τα Clusters, ουσιαστικά είναι η αποθήκευση σε ίδιες σελίδες στον δίσκο όσων περισσότερων πλειαδών του indexing είναι εφικτό να γίνει. Να σημειώθει ότι μετά από κάποια ενημέρωση στο table χρείαζεται να ξαναγίνει cluster, ώστε να διατηρείται οργανωμένη η μνήμη. Όποτε με λιγα λογια ο σκοπός του Cluster είναι να μειώσουμε τον συνολικό αριθμό των disk accessing. Ακόμη, το hash δεν υφίσταται Cluster. Αυτό καθώς έχει πολυπλοκότητα O(1), και η μόνη χρήση του είναι για filtering point queries. Έτσι σε μια συγκεκριμένη αναζήτηση, με η χωρίς κάποιο cluster θα πηγαίνει πάντα στην ίδια page στο δίσκο. Για αυτό τον λόγο λοιπόν δεν γίνεται το cluster σε hash μεθόδους indexing.



(a) (b) (c)

Figure 9: hash(roomtype) and btree(checkout) and CLUSTER checkout

Εχοντας λοιπον τα καλυτερα αποτελεσματα μεχρι στιγμης, με ενα hash index στο roomtype και ενα btree index στο checkout επιλεγουμε να κανουμε clustering στο checkout. Ο μέσος χρόνος εκτέλεσης έμεινε περίπου ίδιος, 15-25 ms. Παρόλα αυτά αυτό που μειώθηκε αισθητά είναι τα heap blocks, από 5895 σε 617.

(a) (b) (c)

Figure 10: btree(checkout) and btree(roomtype) and CLUSTER checkout and CLUSTER roomtype

Στην συνέχεια χωρίς να πειράξουμε το checkout, κάνουμε btree indexing στο roomtype με cluster. Από τα actual time της προηγούμενης περίπτωσης και τα actual time αυτής για το filtering στο roomtype παρατηρούμε ότι το hash είναι πιο γρήγορο από το btree. Αυτό είναι κάτι που περιμέναμε, καθώς το hash έχει πολυπλοκότητα O(1) πάντα σε point query ένω το btree O(logN), το cluster δεν μείωνει δηλαδή την πολυπλοκότητα του btree καθώς O(logN) είναι η ελάχιστη που μπορεί να έχει. Ουσιαστικά το cluster βοηθά στην μείωση του συντελεστή της πολυπλοκότητας των btree indexing, με μεγαλύτερη μείωση στην των εκτέλεση range queries από ότι στα point query. Εδω τα heap blocks απο 14 μειωθηκαν σε 2 και ετσι μειωθηκε ελαχιστα ο χρονος εκτελεσης απο αυτου χωρις clustering. Ο μέσος χρόνος εκτέλεσης σε αυτή την περίπτωση είναι 20-30 ms.

Joins

Για τα joins, πήραμε το καλύτερο συνδιασμό με βάση τα παραπάνω. Δηλαδή btree indexing με cluster στο checkout και hash indexing στο roomtype. Όσο αφορά τα joins, από θεωρία ξέρουμε ότι υπάρχουν 3 διαφορετικά joins: το hash join, το merge join και το nested loop join. Γενικά από θεωρία αν συγκρίνουμε την απόδοση των join μεταξύ τους γνωρίζουμε ότι το nested loop join είναι το καλύτερο join μόνο όταν κάποιο από τα table που θέλουμε να γίνει join έχει πολύ λίγες γραμμές και το άλλο table έχει πολλές γραμμές και indexed στα join columns του. Όσο αφορά το merge join είναι πιο γρήγορο από όλα μόνο όταν το join γίνεται σε δύο αρκετά μεγάλα table, τα οποία έχουν γίνει sorted στο column που γίνεται το join. Ώστοσο αν τα δύο table έχουν σημαντικά μεγάλη διαφορά στο size τους τότε η απόδοση του merge join μειώνεται αρκετά. Γενικότερα σε όλες τις άλλες περιπτώσεις το καλύτερο join σε χρονικη πολυπλοκοτητα είναι το hash join. Ώστόσο, το hash join έχει υψηλότερο κόστος σε κατανάλωση μνήμης και χρήσης του δίσκου δηλαδή χειρότερη χωρική πολυπλοκότητα.

(a)

(b)

(c)

Figure 11: hash(roomtype) and btree(checkout) and CLUSTER checkout and hashjoin=off

Σε αυτη την περιπτωση, εχοντας το hash join disabled παρατηρουμε οτι ο βελτιστοποιητης επιλεγει να κανει στην θεση του join το αμεσως λιγοτερο χρονοβορο join, δηλαδη ενα merge join. Οποτε συμπεραίνουμε ότι παρόλο που τα 2 table έχουν αρκετά μεγάλη διαφορά στο μέγεθος, το room table δεν είναι τόσο μικρό ώστε ο βελτιστοποιητής να διαλέξει το nested loop join. Μέσος χρόνος εκτέλεσης ειναι 20-22 ms.

(a)

(b)

(c)

Figure 12: hash(roomtype) and btree(checkout) and CLUSTER checkout and hashjoin=off and mergejoin=off

Τέλος, αφαιρώντας και το merge join από τις επιλογές του βελτιστοποιητή, προφανώς διαλέγει nested loop. Από τους actual time χρόνους βλέπουμε κίολας γιατι ο βελτιστοποιητής επέλεγε με αυτή τη σειρά τα join για το Query που τρέχαμε. Μέσος χρόνος εκτέλεσης 25-30 ms.