Πανεπιστήμιο Θεσσαλίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Πρόγραμμα Μεταπτυχιακών Σπουδών

Προχωρημένα Θέματα Βάσεων Δεδομένων

Πρακτική Εργασία

Διδάσκων: Βασιλακόπουλος Μιχαήλ

Λορέντζος Δημήτριος – Σωτήριος , 421

Εαρινό 2014

Περιεχόμενα

1. Επιλογή θέματος.....3

2. Περιγραφή και προδιαγραφές του θέματος.....3

3. Δημιουργία Διαγράμματος Οντοτήτων-Συσχετίσεων (ΔΟΣ) με το Microsoft Visio ή το DIA,ή άλλο σχεδιαστικό λογισμικό.....3

4. Μετατροπή ΔΟΣ σε Σχεσιακό Σχήμα (ΣΣ), με έμφαση στους περιορισμούς ακεραιότητας.....3

5. Υλοποίηση του ΣΣ στην PostgreSQL ή σε άλλο Σχεσιακό ΣΔΒΔ της επιλογής σας, χωρίς τον ορισμό ευρετηρίων.....6

6. Εγκατάσταση και ρυθμίσεις στο pgadmin III.....6

7. Εύρεση ή δημιουργία δεδομένων και φόρτωσή τους στη ΒΔ.....6

8. Προσδιορισμός χρήσιμων ερωτημάτων σε SQL επί ενός και επί περισσότερων πινάκων.....6

9. Εκτέλεση των ερωτημάτων για διάφορα πλήθη δεδομένων, μελέτη και καταγραφή του τρόπου και των χρόνων υπολογισμού τους με χρήση της EXPLAIN ή άλλων κατάλληλων εντολών.....9

10. Ορισμός ευρετηρίων για τους πίνακες της ΒΔ.....25

11. Επανάληψη του βήματος 9 και συγκριτική αντιπαράθεση των μεθόδων υπολογισμού και των επιδόσεων για τις δύο περιπτώσεις.....26

12. Συμπεράσματα.....44

13. Λίστα παραδοτέων αρχείων.....44

14. Αναφορές.....45

1. Επιλογή Θέματος

Εφαρμογή κρατήσεων και πελατολόγιου ξενοδοχειακής μονάδας.

2. Περιγραφή και προδιαγραφές του θέματος

Στη παρούσα πρακτική εργασία δημιουργήθηκε μία βάση δεδομένων για τη διαχείριση των πελατών, των κρατήσεων , των δωματίων και των πληρωμών ενός ξενοδοχείου. Η βάση δεδομένων έχει γραφεί στη PostgreSQL με τη βοήθεια του pgAdmin III και έχουν εισαχθεί διάφορες ποσότητες δεδομένων που θα περιγραφούν στη συνέχεια.
Η εφαρμογή χρησιμοποιείται από τους διαχειριστές συστήματος (Πίνακας SystemAdmin) του ξενοδοχείου στην υποδοχή όπου και προσέρχονται οι ενδιαφερόμενοι (Πίνακας Customer) ώστε να δώσουν τα προσωπικά τους στοιχεία και αφού καταγραφούν αυτά, κάνουν κράτηση (Πίνακας Reservation) δωματίου (Πίνακας Room) για συγκεκριμένο χρονικό διάστημα. Πέρα από αυτή την επιλογή η κράτηση μπορεί να καταγραφεί από τους διαχειριστές συστήματος και τηλεφωνικά ,εισάγοντας τα κατάλληλα πεδία στη βάση δεδομένων. Οι πελάτες (Πίνακας Customer) μπορούν επίσης να πληρώσουν (Πίνακας Payment) για τη διαμονή τους στην αρχή ή στο τέλος αυτής. Οι διαχειριστές συστήματος εισάγονται από ένα κεντρικό διαχειριστή (Πίνακας Administrator). Οι διαχειριστές του συστήματος αναλαμβάνουν να εισάγουν τους νέους πελάτες και να ενημερώνουν το σύστημα όταν ένα δωμάτιο γίνει και πάλι διαθέσιμο. Από τη παρούσα βάση δεδομένων μπορούμε να εξάγουμε στατιστικά στοιχεία σχετικά με τις κρατήσεις που έκανε ένας από τους διαχειριστές συστήματος , τη προτίμηση των πελατών σε συγκεκριμένου τύπου δωμάτια (μονόκλινα,δίκλινα,κλπ) και άλλα πολύ χρήσιμα ερωτήματα που έχουν καταγραφεί παρακάτω.

3. Δημιουργία Διαγράμματος Οντοτήτων-Συσχετίσεων (ΔΟΣ) με το Microsoft Visio ή το DIA,ή άλλο σχεδιαστικό λογισμικό.

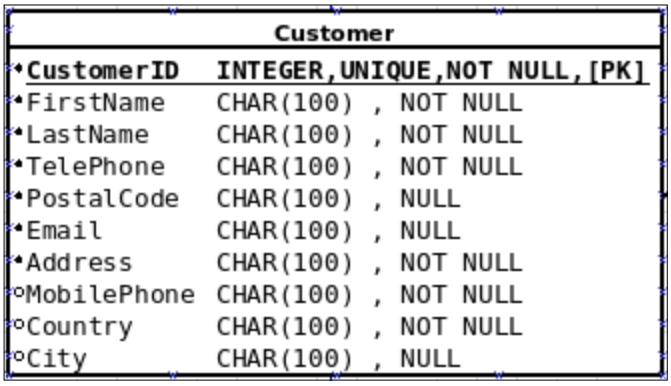
Το διάγραμμα Οντοτήτων – Συσχετίσεων σχεδιάστηκε με το σχεδιαστικό λογισμικό DIA και είναι διαθέσιμο στα παραδοτέα αρχεία.

4. Μετατροπή ΔΟΣ σε Σχεσιακό Σχήμα (ΣΣ), με έμφαση στους περιορισμούς ακεραιότητας.

Το Σχεσιακό Σχήμα επίσης σχεδιάστηκε με το σχεδιαστικό λογισμικό DIA και είναι διαθέσιμο στα παραδοτέα αρχεία.

Οι πίνακες που απαρτίζουν τη βάση δεδομένων είναι:

Ο πίνακας Customer



Ο πίνακας Customer περιέχει όλα τα απαραίτητα στοιχεία των πελατών όπως φαίνονται στο πίνακα παραπάνω.
CustomerID: πρόκειται για το πρωτεύον κλειδί του πίνακα το οποίο για κάθε νέα εισαγωγή εγγραφής αυξάνεται αυτόματα.
FirstName:είναι το όνομα του πελάτη.
LastName:είναι το επώνυμο του πελάτη.
TelePhone:πρόκειται για το σταθερό τηλέφωνο του πελάτη.
PostalCode:είναι ο ταχυδρομικός κώδικας του πελάτη.
Email:είναι το e-mail του πελάτη.
Address:είναι η διεύθυνση του πελάτη.
MobilePhone:είναι το κινητό τηλέφωνο του πελάτη.
Country:πρόκειται για τη χώρα που βρίσκεται η διεύθυνση που μας έδωσε ο πελάτης.
City:πρόκειται για τη πόλη που βρίσκεται η διεύθυνση που μας έδωσε ο πελάτης.

Ο πίνακας Room

Room	
RoomID	INTEGER,UNIQUE,NOT NULL,[PK]
RoomType	CHAR(1) , NOT NULL
Price	REAL , NOT NULL
RoomName	CHAR(5) , UNIQUE , NOT NULL
Booked	SMALLINT , NOT NULL

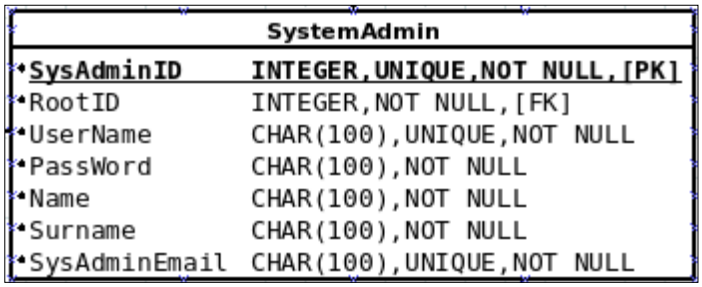
Ο πίνακας Room περιέχει όλη την απαραίτητη πληροφορία σχετικά με τα δωμάτια του ξενοδοχείου. Πιο συγκεκριμένα περιλαμβάνει τα παρακάτω πεδία:
RoomID:Το πρωτεύον κλειδί του πίνακα το οποίο για κάθε νέα εισαγωγή εγγραφής αυξάνεται αυτόματα.
RoomType:Το συγκεκριμένο πεδίο παίρνει τιμές 1,2,3&4 και μας δείχνει αν ένα δωμάτιο είναι μονόκλινο , δίκλινο κλπ.
Price:Αυτό το πεδίο περιέχει τη τιμή του εκάστοτε δωματίου.
RoomName:Αντιπροσωπεύει την ονομασία κάθε δωματίου
Τέλος το πεδίο ***Booked:*** μας δείχνει αν ένα δωμάτιο έχει γίνει κράτηση παίρνοντας τη τιμή 1 και αν δεν εχει γίνει κράτηση παίρνει τη τιμή 0.

Ο πίνακας Administrator

Administrator	
RootID	INTEGER,UNIQUE,NOT NULL,[PK]
RootUsername	CHAR(100),UNIQUE,NOT NULL
RootPassword	CHAR(200) , NOT NULL

Ο πίνακας Administrator περιέχει τις πληροφορίες σχετικά με τον Super User του συστήματος που μπορεί να εισάγει τους διαχειριστές συστήματος και γενικά μπορεί να έχει πρόσβαση σε ολόκληρη την εφαρμογή και να βλέπει για παράδειγμα ποιοι πελάτες έκαναν τις περισσότερες κρατήσεις κ.α. Αυτός ο πίνακας περιλαμβάνει τα παρακάτω πεδία:
RootID:είναι το πρωτεύον κλειδί του πίνακα και αυξάνεται αυτόματα εάν υπάρχουν παραπάνω από ένας super user. Στη παρούσα εργασία έχει εισαχθεί ένας super user.
RootUsername:αποτελεί το όνομα χρήστη του super user.
RootPassword * :είναι ο κωδικός του super user.
* Στη παρούσα εργασία έχει εισαχθεί ο κωδικός χωρίς τη χρήση μονόδρομης συνάρτησης κατακερματισμού για την κρυπτογράφηση και την περεταίρω ασφάλεια του συνθηματικού. Σε πραγματικές συνθήκες θα γινόταν χρήση one way hash function.

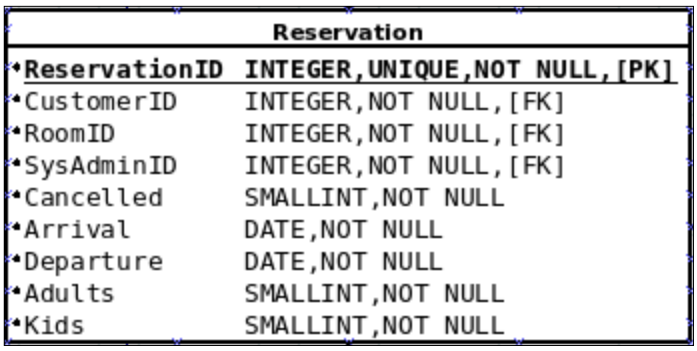
Ο πίνακας SystemAdmin



SystemAdmin	
SysAdminID	INTEGER,UNIQUE,NOT NULL,[PK]
RootID	INTEGER,NOT NULL,[FK]
UserName	CHAR(100),UNIQUE,NOT NULL
PassWord	CHAR(100),NOT NULL
Name	CHAR(100),NOT NULL
Surname	CHAR(100),NOT NULL
SysAdminEmail	CHAR(100),UNIQUE,NOT NULL

Ο πίνακας SystemAdmin περιέχει τους διαχειριστές του συστήματος και τις πληροφορίες που απαρτίζουν τον κάθε ένα ξεχωριστά. Τα πεδία του πίνακα είναι:
SysAdminID:είναι το πρωτεύον κλειδί του πίνακα που αυξάνεται αυτόματα εάν υπάρχουν παραπάνω από ένας διαχειριστές συστήματος.
RootID:είναι το ξένο κλειδί του πίνακα Administrator που δείχνει ποιος super user εισήγαγε το διαχειριστή συστήματος.
UserName:πρόκειται για το όνομα χρήστη του διαχειριστή συστήματος.
PassWord * : είναι ο κωδικός του διαχειριστή συστήματος.
Name:είναι το όνομα του διαχειριστή συστήματος.
Surname:είναι το επώνυμο του διαχειριστή συστήματος.
SysAdminEmail:είναι το e-mail του διαχειριστή συστήματος.
* Ομοίως έχει εισαχθεί ο κωδικός χωρίς τη χρήση μονόδρομης συνάρτησης κατακερματισμού για την κρυπτογράφηση και την περεταίρω ασφάλεια του συνθηματικού. Σε πραγματικές συνθήκες θα γινόταν χρήση one way hash function.

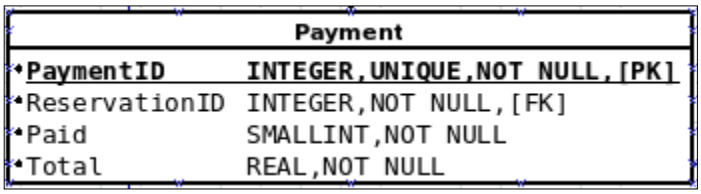
Ο πίνακας Reservation



Reservation	
ReservationID	INTEGER,UNIQUE,NOT NULL,[PK]
CustomerID	INTEGER,NOT NULL,[FK]
RoomID	INTEGER,NOT NULL,[FK]
SysAdminID	INTEGER,NOT NULL,[FK]
Cancelled	SMALLINT,NOT NULL
Arrival	DATE,NOT NULL
Departure	DATE,NOT NULL
Adults	SMALLINT,NOT NULL
Kids	SMALLINT,NOT NULL

Ο πίνακας Reservation περιέχει τις πληροφορίες για τις κρατήσεις των πελατών. Πιο συγκεκριμένα περιλαμβάνει τα πεδία:
ReservationID:είναι το πρωτεύον κλειδί του πίνακα που αυξάνεται αυτόματα.
CustomerID:πρόκειται για το ξένο κλειδί από το πίνακα Customer που χρησιμοποιείται για αναφορά του πελάτη που έκανε μία ή περισσότερες κρατήσεις.
RoomID:πρόκειται για το ξένο κλειδί από το πίνακα Room που χρησιμοποιείται για αναφορά του δωματίου που έγινε η κράτηση.
SysAdminID:είναι το ξένο κλειδί από το πίνακα SystemAdmin που χρησιμοποιείται για αναφορά του διαχειριστή συστήματος που ολοκλήρωσε τη κράτηση.
Cancelled:παίρνει τιμή 1 εάν ακυρώθηκε η κράτηση και 0 εάν δεν ακυρώθηκε.
Arrival:είναι η ημερομηνία άφιξης.
Departure:είναι η ημερομηνία αναχώρησης.
Adults:είναι ο αριθμός των ενηλίκων.
Kids:είναι ο αριθμός των παιδιών.

Ο πίνακας Payment



Payment	
PaymentID	INTEGER,UNIQUE,NOT NULL,[PK]
ReservationID	INTEGER,NOT NULL,[FK]
Paid	SMALLINT,NOT NULL
Total	REAL,NOT NULL

Ο πίνακας Payment περιλαμβάνει τις πληροφορίες σχετικά με την πληρωμή κάθε κράτησης. Τα πεδία του πίνακα είναι:
PaymentID:είναι το πρωτεύον κλειδί του πίνακα που αυξάνεται αυτόματα.
ReservationID:είναι το ξένο κλειδί του πίνακα Reservation που χρησιμοποιείται για αναφορά στη πληρωμή ή όχι μίας κράτησης.
Paid:λαμβάνει τιμή 0 όταν η κράτηση δεν έχει πληρωθεί και 1 όταν η κράτηση έχει πληρωθεί. Ο πελάτης μπορεί και να προπληρώσει τη κράτησή του.
Total:πρόκειται για το συνολικό ποσό που περιλαμβάνει η διαμονή στο ξενοδοχείο και θα πρέπει να καταβάλει ο πελάτης.

5. Υλοποίηση του ΣΣ στην PostgreSQL ή σε άλλο Σχεσιακό ΣΔΒΔ της επιλογής σας, χωρίς τον ορισμό ευρετηρίων.

Η βάση δεδομένων υλοποιήθηκε στην PostgreSQL αρχικά χωρίς τη χρήση ευρετηρίων και βρίσκεται στο αρχείο hotel.sql .

6. Εγκατάσταση και ρυθμίσεις στο pgadmin III.

Η εγκατάσταση και οι ρυθμίσεις στο pgadmin III βρίσκονται στο αρχείο pgadmin_guide_for_db.pdf .

7. Εύρεση ή δημιουργία δεδομένων και φόρτωσή τους στη ΒΔ.

Τα δεδομένα για τη παρούσα πρακτική εργασία τα παρήγαγα με τη βοήθεια του generatedata.com και αφού έκανα clone από το GitHub https://github.com/benkeen/generatedata τοπικά στον Η/Υ μου το project. Το project αυτό απαιτεί να υπάρχει προεγκατεστημένος ο Apache Web Server καθώς επίσης και η MySQL. Στη συνέχεια και αφού είχα δημιουργήσει το σχεσιακό μοντέλο της βάσης δεδομένων προχώρησα στη παραγωγή δεδομένων που υπάρχουν στα παραδοτέα αρχεία. Τέλος έγινε παραγωγή διάφορων ποσοτήτων αρχείων που θα περιγραφούν στη συνέχεια της εργασίας.

8. Προσδιορισμός χρήσιμων ερωτημάτων σε SQL επί ενός και επί περισσότερων πινάκων.

1)Εμφάνισε όλες τις πληροφορίες των δωματίων που είναι διαθέσιμα.

```
SELECT *
FROM Room
WHERE Booked=0 ;
```

2)Εμφάνισε τον αριθμό των δίκλινων δωματίων που είναι διαθέσιμα(ομοίως για μονόκλινα, τρίκλινα,τετράκλινα).

```
SELECT COUNT(Room.RoomName)
FROM Room
WHERE Booked=0 AND RoomType='2';
```

3)Εμφάνισε τον αριθμό των πελατών που έκαναν κράτηση στο ξενοδοχείο χωρίς αυτούς που ακύρωσαν τη κράτησή τους.

```
SELECT COUNT(Reservation.CustomerID)
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=0;
```

4)Εμφάνισε το όνομα, το επώνυμο και το τηλέφωνο των πελατών που ακύρωσαν την κράτησή τους.

```
EXPLAIN ANALYZE SELECT FirstName,LastName,TelePhone
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=1;
```

5)Εμφάνισε το όνομα κάθε δωματίου και τον αριθμό συχνότητας της κράτησης κάθε δωματίου με βάση τον αριθμό συχνότητας κράτησης. (Σημείωση:Για να είναι ένα δωμάτιο αρκετά επισκέψιμο θα πρέπει ο αριθμός συχνότητας επισκέψεως να είναι τουλάχιστον 30 επισκέψεις)

```
SELECT RoomName,COUNT(Room.RoomName) AS visits
FROM Room,Reservation
WHERE Room.RoomID=Reservation.RoomID
GROUP BY Room.RoomName
HAVING COUNT(Room.RoomName)>30;
```

6)Εμφάνισε το όνομα , το επώνυμο και το τηλέφωνο των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν.

```
SELECT FirstName,LastName,TelePhone
FROM Customer,Reservation,Payment
WHERE Customer.CustomerID=Reservation.CustomerID AND
Reservation.ReservationID=Payment.ReservationID AND Cancelled =1 AND Paid=1;
```

7)Εμφάνισε όλες τις πληροφορίες των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν ταξινομημένα ώς προς το συνολικό ποσό σε φθίνουσα σειρά.

```
SELECT *
FROM Customer,Reservation,Payment
WHERE Customer.CustomerID=Reservation.CustomerID AND
Reservation.ReservationID=Payment.ReservationID AND cancelled =1 AND paid=1
ORDER BY Total DESC;
```

8)Εμφάνισε το όνομα , το επώνυμο και την ημερομηνία που οι πελάτες ήρθαν στο ξενοδοχείο κατά το μήνα Ιανουάριο(Δεν ακύρωσαν τη κράτησή τους).

```
SELECT Customer.FirstName,Customer.LastName , Reservation.Arrival
FROM Customer,Reservation
WHERE Customer.CustomerID=Reservation.CustomerID AND Arrival BETWEEN '01/01/2013' AND
'31/01/2013' AND Cancelled=0;
```

9)Εμφάνισε όλες τις πληροφορίες του πελάτη που έδωσε στο ξενοδοχείο το μεγαλύτερο ποσό κατά τη διαμονή του.

```
SELECT *
FROM Customer
WHERE Customer.CustomerID=(SELECT Reservation.CustomerID
FROM Reservation
WHERE Cancelled=0 AND Reservation.ReservationID=(SELECT Payment.ReservationID
FROM Payment
WHERE Total=(SELECT MAX(Total)
FROM Payment)));
```

10)Εμφάνισε το όνομα και το επώνυμο των πελατών που ακύρωσαν την άφιξή τους στο ξενοδοχείο κατά το μήνα Ιανουάριο.

```
SELECT Customer.FirstName,Customer.LastName
FROM Customer,Reservation
WHERE Customer.CustomerID=Reservation.CustomerID AND DATE_PART('month',Arrival)=1 AND
Cancelled=1;
```

11)Εμφάνισε το όνομα του δωματίου που οι πελάτες έκαναν τις περισσότερες κρατήσεις.

```
SELECT Room.RoomName
FROM Room
WHERE Room.RoomID=(
SELECT T.RoomID
FROM(SELECT  Reservation.RoomID,COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) AS T
WHERE K=(SELECT MAX(K)
FROM(SELECT COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) l)
GROUP BY T.RoomID);
```

12)Εμφάνισε το συνολικό αριθμό αφίξεων,για κάθε μήνα,που έγιναν το έτος 2013 σε αύξουσα σειρά με βάση τον μήνα.

```
SELECT DATE_PART('month',Arrival),COUNT(ReservationID) AS NumOfReservations
FROM Reservation
WHERE DATE_PART('year',Arrival)=2013
GROUP BY DATE_PART('month',Arrival)
ORDER BY DATE_PART('month',Arrival);
```

13) Εμφάνισε τον αριθμό των κρατήσεων που έγιναν από την Ελλάδα.

```
SELECT COUNT(ReservationID)
FROM Reservation
INNER JOIN Customer
ON Customer.CustomerID=Reservation.CustomerID
WHERE Country='Greece';
```

14)Ακύρωσε όλες τις κρατήσεις που πρόκειται να αφιχθούν τον Ιανουάριο 2014.

```
UPDATE Reservation
SET Cancelled=1
WHERE DATE_PART('month',Arrival)=1 AND DATE_PART('year',Arrival)=2014;
```

9. Εκτέλεση των ερωτημάτων για διάφορα πλήθη δεδομένων, μελέτη και καταγραφή του τρόπου και των χρόνων υπολογισμού τους με χρήση της EXPLAIN ή άλλων κατάλληλων εντολών.

Τα ερωτήματα που εκτελέστηκαν αρχικά χωρίς τη χρήση ευρετηρίων στη παρούσα βάση δεδομένων ήταν για τους πίνακες με συνολικές εγγραφές ο κάθε ένας ξεχωριστά ως εξής:

1η προσέγγιση
Πίνακας Administrator:1 εγγραφή
Πίνακας SystemAdmin:10 εγγραφές
Πίνακας Customer:300000 εγγραφές
Πίνακας Payment:300000 εγγραφές
Πίνακας Reservation:300000 εγγραφές
Πίνακας Room:1000 εγγραφές

2η προσέγγιση
Πίνακας Administrator:1 εγγραφή
Πίνακας SystemAdmin:10 εγγραφές
Πίνακας Customer:1000000 εγγραφές
Πίνακας Payment:1000000 εγγραφές
Πίνακας Reservation:1000000 εγγραφές
Πίνακας Room:1000 εγγραφές

1)Εμφάνισε όλες τις πληροφορίες των δωματίων που είναι διαθέσιμα.

```
EXPLAIN ANALYZE SELECT *
FROM Room
WHERE Booked=0 ;
```

1η προσέγγιση

Seq Scan on room (cost=0.00..19.50 rows=510 width=18) (actual time=0.016..0.214 rows=510 loops=1)
* Filter: (booked = 0)*

Startup Cost: 0.00
Total Cost:19.50
Plan Rows:510
Return Rows:510
Plan Width:18
Startup time: 0.016 ms
Max time: 0.214 ms
Total runtime: 0.275 ms

Παρατηρούμε πως γίνεται σειριακό σκανάρισμα στο πίνακα Room και φιλτράρισμα των αποτελεσμάτων με βάση τη συνθήκη Booked=0.

2η προσέγγιση
Δεν παίζει ρόλο να υπολογίσουμε το ερώτημα μιας και οι εγγραφές δωματίων έχουν παραμείνει ίδιες.

2)Εμφάνισε τον αριθμό των δίκλινων δωματίων που είναι διαθέσιμα(ομοίως για μονόκλινα, τρίκλινα,τετράκλινα).

```
EXPLAIN ANALYZE SELECT COUNT(Room.RoomName)
FROM Room
WHERE Booked=0 AND RoomType='2';
```

1η προσέγγιση

```
Aggregate  (cost=22.33..22.34 rows=1 width=6) (actual time=0.256..0.256 rows=1 loops=1)
  ->  Seq Scan on room  (cost=0.00..22.00 rows=133 width=6) (actual time=0.011..0.240 rows=132
loops=1)
        Filter: ((booked = 0) AND (roomtype = '2'::bpchar))
```

Startup Cost: 22.33
Total Cost:22.34
Plan Rows:1
Return Rows:1
Plan Width:6
Startup time: 0.256 ms
Max time: 0.256 ms
Total runtime: 0.297 ms

Παρατηρούμε πως γίνεται σειριακό σκανάρισμα στο πίνακα Room και αφού ικανοποιηθούν οι συνθήκες τότε μετράει τα δίκλινα δωμάτια.

2η προσέγγιση
Ομοίως με 1η προσέγγιση.

3)Εμφάνισε τον αριθμό των πελατών που έκαναν κράτηση στο ξενοδοχείο χωρίς αυτούς που ακύρωσαν τη κράτησή τους.

```
EXPLAIN ANALYZE SELECT COUNT(Reservation.CustomerID)
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=0;
```

1η προσέγγιση

```
Aggregate  (cost=56368.25..56368.26 rows=1 width=4) (actual time=370.587..370.587 rows=1 loops=1)
  ->  Hash Join  (cost=8427.62..55991.72 rows=150610 width=4) (actual time=78.597..358.334
rows=150475 loops=1)
      Hash Cond: (customer.customerid = reservation.customerid)
      ->  Seq Scan on customer  (cost=0.00..40500.00 rows=300000 width=4) (actual time=0.006..136.569
rows=300000 loops=1)
      ->  Hash  (cost=5956.00..5956.00 rows=150610 width=4) (actual time=77.853..77.853 rows=150475
loops=1)
            Buckets: 4096  Batches: 8  Memory Usage: 668kB
            ->  Seq Scan on reservation  (cost=0.00..5956.00 rows=150610 width=4) (actual
time=0.030..48.393 rows=150475 loops=1)
                  Filter: (cancelled = 0)
```

Startup Cost: 56368.25
Total Cost: 56368.26
Plan Rows:1
Return Rows:1
Plan Width:4
Startup time: 370.587 ms
Max time: 370.587 ms
Total runtime: 370.696 ms

2η προσέγγιση

Aggregate (cost=186479.32..186479.33 rows=1 width=4) (actual time=1780.692..1780.692 rows=1 loops=1)
-> Hash Join (cost=28009.16..185236.49 rows=497133 width=4) (actual time=247.968..1742.369 rows=500530 loops=1)
Hash Cond: (customer.customerid = reservation.customerid)
-> Seq Scan on customer (cost=0.00..135000.00 rows=1000000 width=4) (actual time=0.008..1017.076 rows=1000000 loops=1)
-> Hash (cost=19853.00..19853.00 rows=497133 width=4) (actual time=244.581..244.581 rows=500530 loops=1)
Buckets: 4096 Batches: 32 Memory Usage: 562kB
-> Seq Scan on reservation (cost=0.00..19853.00 rows=497133 width=4) (actual time=0.036..156.679 rows=500530 loops=1)
Filter: (cancelled = 0)

Startup Cost: 186479.32
Total Cost: 186479.33
Plan Rows:1
Return Rows:1
Plan Width:4
Startup time: 1780.692 ms
Max time: 1780.692 ms
Total runtime: 1780.925 ms

Και στις δυο προσεγγίσεις στο ερώτημα 3) γίνεται σειριακό σκανάρισμα στο πίνακα Reservation φιλτράροντας τις εγγραφές με βάση τη συνθήκη Cancelled=0, έπειτα προχωράμε στη δημιουργία ενός προσωρινού ευρετηρίου Hash και αφού έχει γίνει και στο πίνακα Customer σειριακό σκανάρισμα καταλήγουμε να κάνουμε Join τους 2 πίνακες και συγκεκριμένα Hash Join και εν τέλει να γίνεται η μέτρηση των πελατών ώστε να πάρουμε το αποτέλεσμα που θέλουμε. Θα μας επιστραφεί μία γραμμή με τον αριθμό των πελατών που επισκέφθηκαν το ξενοδοχείο χωρίς αυτούς που ακύρωσαν τη κράτησή τους.

4)Εμφάνισε το όνομα, το επώνυμο και το τηλέφωνο των πελατών που ακύρωσαν την κράτησή τους.

```
EXPLAIN ANALYZE SELECT FirstName,LastName,TelePhone
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=1;
```

1η προσέγγιση

Merge Join (cost=20849.86..69518.96 rows=149500 width=303) (actual time=134.633..393.079 rows=149525 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..45303.42 rows=300000 width=307) (actual time=0.012..141.808 rows=299996 loops=1)
-> Materialize (cost=20849.37..21596.87 rows=149500 width=4) (actual time=134.615..173.880 rows=149525 loops=1)
-> Sort (cost=20849.37..21223.12 rows=149500 width=4) (actual time=134.607..155.878 rows=149525 loops=1)
Sort Key: reservation.customerid
Sort Method: external sort Disk: 2048kB
-> Seq Scan on reservation (cost=0.00..5956.00 rows=149500 width=4) (actual time=0.015..57.114 rows=149525 loops=1)
Filter: (cancelled = 1)
Rows Removed by Filter: 150475

Startup Cost: 20849.86
Total Cost: 69518.96
Plan Rows: 149500
Return Rows: 149525
Plan Width: 303
Startup time: 134.633 ms
Max time: 393.079 ms
Total runtime: 400.322 ms

2η προσέγγιση

Merge Join (cost=81304.91..243598.04 rows=503467 width=303) (actual time=405.555..1249.819 rows=499470 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=307) (actual time=0.025..464.241 rows=999999 loops=1)
-> Materialize (cost=81304.20..83821.53 rows=503467 width=4) (actual time=405.524..532.127 rows=499470 loops=1)
-> Sort (cost=81304.20..82562.86 rows=503467 width=4) (actual time=405.504..473.093 rows=499470 loops=1)
Sort Key: reservation.customerid
Sort Method: external sort Disk: 6832kB
-> Seq Scan on reservation (cost=0.00..19853.00 rows=503467 width=4) (actual time=0.029..162.488 rows=499470 loops=1)
Filter: (cancelled = 1)
Rows Removed by Filter: 500530

Startup Cost: 81304.91
Total Cost: 243598.04
Plan Rows: 503467
Return Rows: 499470
Plan Width: 303
Startup time: 405.555 ms
Max time: 1249.819 ms
Total runtime: 1274.595 ms

Στις δυο παραπάνω προσεγγίσεις γίνεται σειριακό σκανάρισμα στο πίνακα Reservation φιλτράροντας τις εγγραφές με βάσει τη συνθήκη Cancelled=1. Έπειτα γίνεται ταξινόμηση σύμφωνα με το πρωτεύον κλειδί. Στη συνέχεια ακολουθεί η διαδικασία Materialize όπου αποθηκεύει τα δεδομένα στη μνήμη και έπειτα τα επιστρέφει σε κάθε ένα ξεχωριστό πέρασμα. Τέλος γίνεται σκανάρισμα στο πίνακα Customer και πραγματοποιείται η πράξη του Join.

5)Εμφάνισε το όνομα κάθε δωματίου και τον αριθμό συχνότητας της κράτησης κάθε δωματίου με βάση τον αριθμό συχνότητας κράτησης. (Σημείωση: Για να είναι ένα δωμάτιο αρκετά επισκέψιμο θα πρέπει ο αριθμός συχνότητας επισκέψεως να είναι τουλάχιστον 30 επισκέψεις)

```
EXPLAIN ANALYZE SELECT RoomName,COUNT(Room.RoomName) AS visits
FROM Room,Reservation
WHERE Room.RoomID=Reservation.RoomID
GROUP BY Room.RoomName
HAVING COUNT(Room.RoomName)>30;
```

1η προσέγγιση

HashAggregate (cost=11610.50..11623.00 rows=1000 width=6) (actual time=176.535..176.721 rows=1000 loops=1)
Filter: (count(room.roomname) > 30)
-> Hash Join (cost=29.50..9360.50 rows=300000 width=6) (actual time=1.496..110.418 rows=300000 loops=1)
Hash Cond: (reservation.roomid = room.roomid)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.318..31.290 rows=300000 loops=1)
-> Hash (cost=17.00..17.00 rows=1000 width=10) (actual time=1.147..1.147 rows=1000 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 43kB
-> Seq Scan on room (cost=0.00..17.00 rows=1000 width=10) (actual time=0.015..0.606 rows=1000 loops=1)

Startup Cost: 11610.50
Total Cost: 11623.00
Plan Rows: 1000
Return Rows:1000
Plan Width:6
Startup time: 176.535 ms
Max time: 176.721 ms
Total runtime: 176.863 ms

2η προσέγγιση

HashAggregate (cost=38632.50..38645.00 rows=1000 width=6) (actual time=585.002..585.138 rows=1000 loops=1)
Filter: (count(room.roomname) > 30)
-> Hash Join (cost=29.50..31132.50 rows=1000000 width=6) (actual time=1.020..364.679 rows=1000000 loops=1)
Hash Cond: (reservation.roomid = room.roomid)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.021..101.822 rows=1000000 loops=1)
-> Hash (cost=17.00..17.00 rows=1000 width=10) (actual time=0.982..0.982 rows=1000 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 43kB
-> Seq Scan on room (cost=0.00..17.00 rows=1000 width=10) (actual time=0.011..0.499 rows=1000 loops=1)

Startup Cost: 38632.50
Total Cost: 38645.00
Plan Rows: 1000
Return Rows:1000
Plan Width:6
Startup time: 585.002 ms
Max time: 585.138 ms
Total runtime: 585.325 ms

Και στις δυο προσεγγίσεις στο ερώτημα 5) γίνεται σειριακό σκανάρισμα στο πίνακα Room έπειτα προχωράμε στη δημιουργία ενός προσωρινού ευρετηρίου Hash και αφού έχει γίνει και στο πίνακα Reservation σειριακό σκανάρισμα καταλήγουμε να κάνουμε Join τους 2 πίνακες και συγκεκριμένα Hash Join.Τέλος πραγματοποιείται η λειτουργία του HashAggregate για τα ομαδοποιημένα αποτελέσματα που έχουν συγκεντρωθεί αφού έχει προηγηθεί η συνθήκη count(room.roomname) > 30 για τις εγγραφές που την ικανοποιούν .Αυτή η διαδικασία (HashAggregate) είναι πολύ πιο γρήγορη από το να ταξινομούμε και να ομαδοποιούμε σε 2 φάσεις. Το αποτέλεσμα θα είναι 1000 εγγραφές.

6)Εμφάνισε το όνομα , το επώνυμο και το τηλέφωνο των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν.

```
EXPLAIN ANALYZE SELECT FirstName,LastName,TelePhone
FROM Customer,Reservation,Payment
WHERE Customer.CustomerID=Reservation.CustomerID AND
Reservation.ReservationID=Payment.ReservationID AND Cancelled =1 AND Paid=1;
```

1η προσέγγιση

Merge Join (cost=24630.75..71987.01 rows=74476 width=303) (actual time=245.150..473.204 rows=74530 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..45303.42 rows=300000 width=307) (actual time=0.024..139.528 rows=299996 loops=1)
-> Materialize (cost=24630.29..25002.67 rows=74476 width=4) (actual time=245.120..282.078 rows=74530 loops=1)
-> Sort (cost=24630.29..24816.48 rows=74476 width=4) (actual time=245.116..272.085 rows=74530 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 1008kB
-> Hash Join (cost=7824.12..17585.01 rows=74476 width=4) (actual time=81.489..207.851 rows=74530 loops=1)
Hash Cond: (reservation.reservationid = payment.reservationid)
-> Seq Scan on reservation (cost=0.00..5956.00 rows=149500 width=8) (actual time=0.017..48.426 rows=149525 loops=1)
Filter: (cancelled = 1)
Rows Removed by Filter: 150475
-> Hash (cost=5372.00..5372.00 rows=149450 width=4) (actual time=81.356..81.356 rows=149323 loops=1)
Buckets: 4096 Batches: 8 Memory Usage: 668kB
-> Seq Scan on payment (cost=0.00..5372.00 rows=149450 width=4) (actual time=0.026..51.276 rows=149323 loops=1)
Filter: (paid = 1)
Rows Removed by Filter: 150677

Startup Cost: 24630.75
Total Cost: 71987.01
Plan Rows: 74476
Return Rows: 74530
Plan Width: 303
Startup time: 245.150 ms
Max time: 473.204 ms
Total runtime: 477.052 ms

2η προσέγγιση

Merge Join (cost=84082.31..241958.33 rows=251045 width=303) (actual time=798.895..1563.708 rows=249383 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=307) (actual time=0.025..460.901 rows=999999 loops=1)
-> Materialize (cost=84081.74..85336.97 rows=251045 width=4) (actual time=798.863..929.983 rows=249383 loops=1)
-> Sort (cost=84081.74..84709.35 rows=251045 width=4) (actual time=798.859..894.843 rows=249383 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 3408kB
-> Hash Join (cost=28113.34..58132.54 rows=251045 width=4) (actual time=255.426..641.801 rows=249383 loops=1)
Hash Cond: (payment.reservationid = reservation.reservationid)
-> Seq Scan on payment (cost=0.00..17906.00 rows=498633 width=4) (actual time=0.057..140.956 rows=498774 loops=1)
Filter: (paid = 1)
Rows Removed by Filter: 501226
-> Hash (cost=19853.00..19853.00 rows=503467 width=8) (actual time=254.943..254.943 rows=499470 loops=1)
Buckets: 4096 Batches: 32 Memory Usage: 618kB
-> Seq Scan on reservation (cost=0.00..19853.00 rows=503467 width=8) (actual time=0.012..159.438 rows=499470 loops=1)
Filter: (cancelled = 1)
Rows Removed by Filter: 500530

Startup Cost: 84082.31
Total Cost: 241958.33
Plan Rows: 251045
Return Rows: 249383
Plan Width: 303
Startup time: 798.895 ms
Max time: 1563.708 ms
Total runtime: 1576.237 ms

Και στις δυο προσεγγίσεις στο ερώτημα 6) γίνεται σειριακό σκανάρισμα στο πίνακα Reservation έπειτα προχωράμε στη δημιουργία ενός προσωρινού ευρετηρίου Hash και αφού έχει γίνει και στο πίνακα Payment σειριακό σκανάρισμα καταλήγουμε να κάνουμε Join τους 2 πίνακες και συγκεκριμένα Hash Join.Έπειτα γίνεται ταξινόμηση με βάση το ξένο κλειδί Reservation.CustomerID καταλήγοντας στη διαδικασία Materialize που περιγράψαμε σε προηγούμενο ερώτημα. Τέλος γίνεται σκανάρισμα στο πίνακα Customer και στη συνέχεια Merge Join.

7)Εμφάνισε όλες τις πληροφορίες των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν ταξινομημένα ώς προς το συνολικό ποσό σε φθίνουσα σειρά.

```
EXPLAIN ANALYZE SELECT *
FROM Customer,Reservation,Payment
WHERE Customer.CustomerID=Reservation.CustomerID AND
Reservation.ReservationID=Payment.ReservationID AND cancelled =1 AND paid=1
ORDER BY Total DESC;
```

1η προσέγγιση

Sort (cost=142192.22..142376.41 rows=73679 width=957) (actual time=819.752..864.450 rows=74530 loops=1)
Sort Key: payment.total
Sort Method: external merge Disk: 70976kB
-> Merge Join (cost=26936.05..74278.65 rows=73679 width=957) (actual time=262.145..518.052 rows=74530 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.00..45303.28 rows=300000 width=913) (actual time=0.017..141.042 rows=299996 loops=1)
-> Materialize (cost=26936.02..27304.41 rows=73679 width=44) (actual time=262.119..308.240 rows=74530 loops=1)
-> Sort (cost=26936.02..27120.22 rows=73679 width=44) (actual time=262.115..294.703 rows=74530 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 4208kB
-> Hash Join (cost=7944.50..18711.45 rows=73679 width=44) (actual time=79.345..207.569 rows=74530 loops=1)
Hash Cond: (reservation.reservationid = payment.reservationid)
-> Seq Scan on reservation (cost=0.00..5956.00 rows=149390 width=30) (actual time=0.011..41.794 rows=149525 loops=1)
Filter: (cancelled = 1)
-> Hash (cost=5372.00..5372.00 rows=147960 width=14) (actual time=79.177..79.177 rows=149323 loops=1)
Buckets: 4096 Batches: 8 Memory Usage: 891kB
-> Seq Scan on payment (cost=0.00..5372.00 rows=147960 width=14) (actual time=0.025..50.000 rows=149323 loops=1)
Filter: (paid = 1)

Startup Cost: 142192.22
Total Cost: 142376.41
Plan Rows: 73679
Return Rows: 74530
Plan Width: 957
Startup time: ms
Max time: ms
Total runtime: 884.238 ms

2η προσέγγιση

Sort (cost=598421.52..599047.38 rows=250344 width=957) (actual time=2921.063..3069.429 rows=249383 loops=1)
Sort Key: payment.total
Sort Method: external merge Disk: 237456kB
-> Merge Join (cost=102364.10..260228.23 rows=250344 width=957) (actual time=918.080..1741.406 rows=249383 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.00..150983.36 rows=1000000 width=913) (actual time=0.005..455.075 rows=999999 loops=1)
-> Materialize (cost=102363.98..103615.70 rows=250344 width=44) (actual time=918.065..1070.687 rows=249383 loops=1)
-> Sort (cost=102363.98..102989.84 rows=250344 width=44) (actual time=918.061..1027.363 rows=249383 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 14128kB
-> Hash Join (cost=29576.84..64509.19 rows=250344 width=44) (actual time=236.023..674.288 rows=249383 loops=1)
Hash Cond: (payment.reservationid = reservation.reservationid)

-> Seq Scan on payment (cost=0.00..17906.00 rows=497833 width=14) (actual time=0.013..134.270 rows=498774 loops=1)
Filter: (paid = 1)
-> Hash (cost=19853.00..19853.00 rows=502867 width=30) (actual time=235.769..235.769 rows=499470 loops=1)
Buckets: 2048 Batches: 32 Memory Usage: 992kB
-> Seq Scan on reservation (cost=0.00..19853.00 rows=502867 width=30) (actual time=0.005..148.352 rows=499470 loops=1)
Filter: (cancelled = 1)

Startup Cost: 598421.52
Total Cost: 599047.38
Plan Rows: 250344
Return Rows: 249383
Plan Width: 957
Startup time: 0.005 ms
Max time: 455.075 ms
Total runtime: 3123.515 ms

Παρόμοια σχεδόν διαδικασία με το ερώτημα 6) παρουσιάζεται στο ερώτημα 7) όπου η διαδικασία είναι ακριβώς η ίδια με προηγουμένως απλώς στο τελευταίο στάδιο πραγματοποιείται ταξινόμηση με βάση το συνολικό πόσο Payment.Total.

8)Εμφάνισε το όνομα , το επώνυμο και την ημερομηνία που οι πελάτες ήρθαν στο ξενοδοχείο κατά το μήνα Ιανουάριο (Δεν ακύρωσαν τη κράτησή τους).

```
EXPLAIN ANALYZE SELECT Customer.FirstName,Customer.LastName , Reservation.Arrival
FROM Customer,Reservation
WHERE Customer.CustomerID=Reservation.CustomerID AND Arrival BETWEEN '01/01/2013' AND
'31/01/2013' AND Cancelled=0;
```

1η προσέγγιση
Nested Loop (cost=0.00..38985.04 rows=4834 width=206) (actual time=0.042..78.853 rows=4813 loops=1)
-> Seq Scan on reservation (cost=0.00..7456.00 rows=4834 width=8) (actual time=0.023..53.698 rows=4813 loops=1)
Filter: ((arrival >= '2013-01-01'::date) AND (arrival <= '2013-01-31'::date) AND (cancelled = 0))
-> Index Scan using customer_id on customer (cost=0.00..6.51 rows=1 width=206) (actual time=0.004..0.005 rows=1 loops=4813)
Index Cond: (customerid = reservation.customerid)
Startup Cost: 0.00
Total Cost: 38985.04
Plan Rows: 4834
Return Rows: 4813
Plan Width: 206
Startup time: 0.042 ms
Max time: 78.853 ms
Total runtime: 79.215 ms

Στο ερώτημα 8) στη πρώτη προσέγγιση ο Optimizer επιλέγει σε σειριακή αναζήτηση στο πίνακα Reservation κι εδώ έρχεται και το Nested Loop που χρησιμοποιείται όταν κάνουμε Join 2 πίνακες ώστε να τροφοδοτήσουμε τα φιλτραρισμένα κριτήρια στο ευρετήριο που σκανάρει για να βρει τις εγγραφές που ικανοποιούν τη συνθήκη.

2η προσέγγιση

Merge Join (cost=27047.23..180973.81 rows=29550 width=206) (actual time=174.461..384.158 rows=31093 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.00..150983.36 rows=1000000 width=206) (actual time=0.020..162.103 rows=400000 loops=1)
-> Sort (cost=27047.22..27121.09 rows=29550 width=8) (actual time=174.409..179.266 rows=31093 loops=1)
Sort Key: reservation.customerid
Sort Method: external sort Disk: 672kB
-> Seq Scan on reservation (cost=0.00..24853.00 rows=29550 width=8) (actual time=0.047..154.493 rows=31093 loops=1)
Filter: ((arrival >= '2013-01-01'::date) AND (arrival <= '2013-01-31'::date) AND (cancelled = 0))

Startup Cost: 27047.23
Total Cost: 180973.81
Plan Rows: 29550
Return Rows: 31093
Plan Width:206
Startup time: 174.461 ms
Max time: 384.158 ms
Total runtime: 385.918 ms

Στο ερώτημα 8) γίνεται σειριακό σκανάρισμα στο πίνακα Reservation και έπειτα ταξινόμηση με βάση το ξένο κλειδί Reservation.CustomerID. Προχωράμε με το σκανάρισμα του πίνακα Customer και η συνολική διαδικασία ολοκληρώνεται κάνοντας Merge Join.

Σύγκριση περιπτώσεων

Ανάμεσα σε αυτές τις 2 προσεγγίσεις του ερωτήματος 8 παρατηρούμε πως ο Optimizer επέλεξε διαφορετικό πλάνο εκτέλεσης λόγω του όγκου των δεδομένων. Αφού δεν έχουμε ορίσει ευρετήρια ο Optimizer επιλέγει το καλύτερο δυνατό πλάνο με το χαμηλότερο κόστος.

9)Εμφάνισε όλες τις πληροφορίες του πελάτη που έδωσε στο ξενοδοχείο το μεγαλύτερο ποσό κατά τη διαμονή του.

```
EXPLAIN ANALYZE SELECT *
FROM Customer
WHERE Customer.CustomerID=(SELECT Reservation.CustomerID
FROM Reservation
WHERE Cancelled=0 AND Reservation.ReservationID=(SELECT Payment.ReservationID
FROM Payment
WHERE Total=(SELECT MAX(Total)
FROM Payment)));
```

1η προσέγγιση

Index Scan using customer_id on customer (cost=10752.32..10760.62 rows=1 width=913) (actual time=107.046..107.046 rows=1 loops=1)
Index Cond: (customerid = $2)
InitPlan 3 (returns $2)
-> Index Scan using reservation_id on reservation (cost=10744.01..10752.32 rows=1 width=4) (actual time=106.990..106.993 rows=1 loops=1)
Index Cond: (reservationid = $1)
Filter: (cancelled = 0)
InitPlan 2 (returns $1)
-> Seq Scan on payment (cost=5372.01..10744.01 rows=1 width=4) (actual time=104.361..105.198 rows=1 loops=1)
Filter: (total = $0)

InitPlan 1 (returns $0)
-> Aggregate (cost=5372.00..5372.01 rows=1 width=4) (actual time=71.076..71.076 rows=1 loops=1)
-> Seq Scan on payment (cost=0.00..4622.00 rows=300000 width=4) (actual time=0.011..32.220 rows=300000 loops=1)

Startup Cost: 10752.32
Total Cost: 10760.62
Plan Rows:1
Return Rows:1
Plan Width:913
Startup time: 107.046 ms
Max time: 107.046 ms
Total runtime: 107.158 ms

2η προσέγγιση
Index Scan using customer_id on customer (cost=35820.39..35828.77 rows=1 width=913) (actual time=312.832..312.833 rows=1 loops=1)
Index Cond: (customerid = $2)
InitPlan 3 (returns $2)
-> Index Scan using reservation_id on reservation (cost=35812.01..35820.39 rows=1 width=4) (actual time=312.780..312.782 rows=1 loops=1)
Index Cond: (reservationid = $1)
Filter: (cancelled = 0)
InitPlan 2 (returns $1)
-> Seq Scan on payment (cost=17906.01..35812.01 rows=2 width=4) (actual time=227.903..311.289 rows=1 loops=1)
Filter: (total = $0)
InitPlan 1 (returns $0)
-> Aggregate (cost=17906.00..17906.01 rows=1 width=4) (actual time=197.879..197.879 rows=1 loops=1)
-> Seq Scan on payment (cost=0.00..15406.00 rows=1000000 width=4) (actual time=0.004..87.404 rows=1000000 loops=1)

Startup Cost: 35820.39
Total Cost: 35828.77
Plan Rows:1
Return Rows:1
Plan Width:913
Startup time: 312.832 ms
Max time: 312.833 ms
Total runtime: 312.889 ms

Στο ερώτημα 9) και στις 2 προσεγγίσεις παρατηρούμε πως αρχικά γίνεται σειριακό σκανάρισμα στο πίνακα και επιλέγεται το MAX(Total). Χρησιμοποιείται το InitPlan 1 ώστε να υπολογιστεί το subselect και επιστρέφεται μία γραμμή με το MAX(Total). Έπειτα προχωράμε σε σειριακό σκανάρισμα του πίνακα Payment και βρίσκουμε το Payment.ReservationID . Κι εδώ χρησιμοποιείται το InitPlan 2 ώστε να υπολογιστεί το subselect. Τέλος γίνεται ο υπολογισμός του πρώτου subselect με το InitPlan 3 και φιλτράρισμα των εγγραφών με βάση τη συνθήκη μας ώστε να επιστραφεί το Reservation.CustomerID. Τέλος επιστρέφεται μία εγγραφή με τις πληροφορίες του πελάτη.

10)Εμφάνισε το όνομα και το επώνυμο των πελατών που ακύρωσαν την άφιξή τους στο ξενοδοχείο κατά το μήνα Ιανουάριο.

```
EXPLAIN ANALYZE SELECT Customer.FirstName,Customer.LastName
FROM Customer,Reservation
WHERE Customer.CustomerID=Reservation.CustomerID AND DATE_PART('month',Arrival)=1 AND
Cancelled=1;
```

1η προσέγγιση

Nested Loop (cost=0.00..13519.81 rows=747 width=202) (actual time=0.079..89.459 rows=4739 loops=1)
-> Seq Scan on reservation (cost=0.00..8206.00 rows=747 width=4) (actual time=0.059..67.033 rows=4739 loops=1)
Filter: ((cancelled = 1) AND (date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision))
-> Index Scan using customer_id on customer (cost=0.00..7.10 rows=1 width=206) (actual time=0.004..0.004 rows=1 loops=4739)
Index Cond: (customerid = reservation.customerid)

Startup Cost: 0.00
Total Cost: 13519.81
Plan Rows: 747
Return Rows: 4739
Plan Width: 202
Startup time: 0.079 ms
Max time: 89.459 ms
Total runtime: 89.818 ms

2η προσέγγιση

Nested Loop (cost=0.00..47212.96 rows=2514 width=202) (actual time=0.048..472.745 rows=80966 loops=1)
-> Seq Scan on reservation (cost=0.00..27353.00 rows=2514 width=4) (actual time=0.035..242.024 rows=80966 loops=1)
Filter: ((cancelled = 1) AND (date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision))
-> Index Scan using customer_id on customer (cost=0.00..7.89 rows=1 width=206) (actual time=0.002..0.002 rows=1 loops=80966)
Index Cond: (customerid = reservation.customerid)

Startup Cost: 0.00
Total Cost: 47212.96
Plan Rows: 2514
Return Rows: 80966
Plan Width: 202
Startup time: 0.048 ms
Max time: 472.745 ms
Total runtime: 477.377 ms

Στο ερώτημα 10) παρατηρούμε πως ο optimizer ακολουθεί παρόμοιο πλάνο και στις δυο περιπτώσεις. Αρχικά γίνεται σειριακό σκανάρισμα στο πίνακα Reservation και φιλτράρισμα των εγγραφών με βάση τη συνθήκη DATE_PART('month',Arrival)=1 AND Cancelled=1. Έπειτα γίνεται σκανάρισμα στο ευρετήριο στο πίνακα Customer για να γίνουν Join οι πίνακες με βάση το πρωτεύον και το ξένο κλειδί. Όλη η παραπάνω διαδικασία εκτελείται πάνω ένα Nested Loop που εξηγήσαμε προηγουμένως ποια είναι η λειτουργία του.

11)Εμφάνισε το όνομα του δωματίου που οι πελάτες έκαναν τις περισσότερες κρατήσεις.

```
EXPLAIN ANALYZE SELECT Room.RoomName
FROM Room
WHERE Room.RoomID=(
SELECT T.RoomID
FROM(SELECT  Reservation.RoomID,COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) AS T
WHERE K=(SELECT MAX(K)
FROM(SELECT COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) l)
GROUP BY T.RoomID);
```

1η προσέγγιση

Index Scan using room_id on room (cost=14211.51..14219.78 rows=1 width=6) (actual time=176.317..176.318 rows=1 loops=1)
Index Cond: (roomid = $1)
InitPlan 2 (returns $1)
-> HashAggregate (cost=14209.51..14211.51 rows=200 width=4) (actual time=176.295..176.295 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Aggregate (cost=6728.50..6728.51 rows=1 width=8) (actual time=87.663..87.663 rows=1 loops=1)
-> HashAggregate (cost=6706.00..6716.00 rows=1000 width=4) (actual time=87.482..87.596 rows=1000 loops=1)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.006..25.189 rows=300000 loops=1)
-> HashAggregate (cost=7456.00..7468.50 rows=1000 width=4) (actual time=176.249..176.292 rows=1 loops=1)
Filter: (count(*) = $0)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.010..24.558 rows=300000 loops=1)

Startup Cost: 14211.51
Total Cost: 14219.78
Plan Rows:1
Return Rows:1
Plan Width:6
Startup time: 176.317 ms
Max time: 176.318 ms
Total runtime: 176.408 ms

2η προσέγγιση

Index Scan using room_id on room (cost=47255.51..47263.78 rows=1 width=6) (actual time=572.260..572.261 rows=1 loops=1)
Index Cond: (roomid = $1)
InitPlan 2 (returns $1)
-> HashAggregate (cost=47253.51..47255.51 rows=200 width=4) (actual time=572.235..572.235 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Aggregate (cost=22375.50..22375.51 rows=1 width=8) (actual time=281.191..281.191 rows=1 loops=1)
-> HashAggregate (cost=22353.00..22363.00 rows=1000 width=4) (actual time=280.982..281.106 rows=1000 loops=1)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.007..84.550 rows=1000000 loops=1)

-> HashAggregate (cost=24853.00..24865.50 rows=1000 width=4) (actual time=572.227..572.231 rows=1 loops=1)
Filter: (count(*) = $0)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.035..86.939 rows=1000000 loops=1)

Startup Cost: 47255.51
Total Cost: 47263.78
Plan Rows:1
Return Rows:1
Plan Width:6
Startup time: 572.260 ms
Max time: 572.261 ms
Total runtime: 572.484 ms

Στο ερώτημα 11) και στις δυο περιπτώσεις γίνεται σειριακή αναζήτηση στο πίνακα Reservation και στη συνέχεια πραγματοποιείται η λειτουργία του HashAggregate για τα ομαδοποιημένα αποτελέσματα που έχουν συγκεντρωθεί που περιγράψαμε και προηγουμένως.Στη συνέχεια επαναλαμβάνουμε την ίδια διαδικασία φθάνοντας στο σημείο InitPlan 1 (return $0). Το σχέδιο αυτό συμβαίνει κάθε φορά που υπάρχει ένα μέρος του ερωτήματός που πρέπει να υπολογίζεται πριν από οτιδήποτε άλλο, και δεν εξαρτάται από τίποτα στο υπόλοιπο του ερωτήματός μας. Έπειτα πραγματοποιείται πάλι η διαδικασία του HashAggregate και λαμβάνουμε πάλι αποτελέσματα.

12)Εμφάνισε το συνολικό αριθμό αφίξεων ,για κάθε μήνα,που έγιναν το έτος 2013 σε αύξουσα σειρά με βάση τον μήνα.

```
EXPLAIN ANALYZE SELECT DATE_PART('month',Arrival),COUNT(ReservationID) AS
NumOfReservations
FROM Reservation
WHERE DATE_PART('year',Arrival)=2013
GROUP BY DATE_PART('month',Arrival)
ORDER BY DATE_PART('month',Arrival);
```

1η προσέγγιση
Sort (cost=7471.04..7471.05 rows=2 width=8) (actual time=243.633..243.634 rows=12 loops=1)
Sort Key: (date_part('month'::text, (arrival)::timestamp without time zone))
Sort Method: quicksort Memory: 25kB
-> HashAggregate (cost=7471.00..7471.03 rows=2 width=8) (actual time=243.594..243.597 rows=12 loops=1)
-> Seq Scan on reservation (cost=0.00..7463.50 rows=1500 width=8) (actual time=0.044..179.409 rows=300000 loops=1)
Filter: (date_part('year'::text, (arrival)::timestamp without time zone) = 2013::double precision)

Startup Cost: 7471.04
Total Cost: 7471.05
Plan Rows:2
Return Rows:12
Plan Width:8
Startup time: 243.633 ms
Max time: 243.634 ms
Total runtime: 243.708 ms

2η προσέγγιση

Sort (cost=24903.07..24903.08 rows=3 width=8) (actual time=576.205..576.205 rows=12 loops=1)
Sort Key: (date_part('month'::text, (arrival)::timestamp without time zone))
Sort Method: quicksort Memory: 25kB
-> HashAggregate (cost=24903.00..24903.04 rows=3 width=8) (actual time=576.166..576.169 rows=12 loops=1)
-> Seq Scan on reservation (cost=0.00..24878.00 rows=5000 width=8) (actual time=0.050..457.847 rows=600000 loops=1)
Filter: (date_part('year'::text, (arrival)::timestamp without time zone) = 2013::double precision)

Startup Cost: 24903.07
Total Cost: 24903.08
Plan Rows:3
Return Rows:12
Plan Width:8
Startup time: 576.205 ms
Max time: 576.205 ms
Total runtime: 576.283 ms

Στο ερώτημα 12) και στις δυο περιπτώσεις αρχικά γίνεται σειριακό σκανάρισμα στο πίνακα Reservation και στη συνέχεια ο πίνακας γίνεται Hash όπου τα Buckets περιέχουν δεδομένα. Παράλληλα έχει γίνει σειριακό σκανάρισμα στο πίνακα Payment για να ικανοποιηθεί και η συνθήκη Paid<>0 . Τέλος γίνεται Hash Join στους 2 πίνακες και αθροίζονται τα ποσά ώστε να πάρουμε ως αποτέλεσμα μια γραμμή με το αποτέλεσμα που ζητάμε.

13) Εμφάνισε τον αριθμό των κρατήσεων που έγιναν από την Ελλάδα.

```
EXPLAIN ANALYZE SELECT COUNT(ReservationID)
FROM Reservation
INNER JOIN Customer
ON Customer.CustomerID=Reservation.CustomerID
WHERE Country='Greece';
```

1η προσέγγιση

Aggregate (cost=47611.12..47611.14 rows=1 width=4) (actual time=837.350..837.350 rows=1 loops=1)
-> Hash Join (cost=41265.06..47608.11 rows=1205 width=4) (actual time=764.733..837.180 rows=1159 loops=1)
Hash Cond: (reservation.customerid = customer.customerid)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=8) (actual time=0.375..36.780 rows=300000 loops=1)
-> Hash (cost=41250.00..41250.00 rows=1205 width=4) (actual time=764.205..764.205 rows=1159 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 41kB
-> Seq Scan on customer (cost=0.00..41250.00 rows=1205 width=4) (actual time=0.374..762.625 rows=1159 loops=1)
Filter: (country = 'Greece'::bpchar)
Rows Removed by Filter: 298841

Startup Cost: 47611.12
Total Cost: 47611.14
Plan Rows:1
Return Rows:1
Plan Width:4
Startup time: 837.350 ms
Max time: 837.350 ms
Total runtime: 837.519 ms

2η προσέγγιση

Aggregate (cost=162453.40..162453.41 rows=1 width=4) (actual time=2791.833..2791.833 rows=1 loops=1)
-> Hash Join (cost=137550.20..162443.36 rows=4016 width=4) (actual time=2547.061..2791.331 rows=3924 loops=1)
Hash Cond: (reservation.customerid = customer.customerid)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=8) (actual time=0.252..98.154 rows=1000000 loops=1)
-> Hash (cost=137500.00..137500.00 rows=4016 width=4) (actual time=2546.740..2546.740 rows=3924 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 138kB
-> Seq Scan on customer (cost=0.00..137500.00 rows=4016 width=4) (actual time=870.691..2545.173 rows=3924 loops=1)
Filter: (country = 'Greece'::bpchar)
Rows Removed by Filter: 996076

Startup Cost: 162453.40
Total Cost: 162453.41
Plan Rows:1
Return Rows:1
Plan Width:4
Startup time: 2791.833 ms
Max time: 2791.833 ms
Total runtime: 2791.906 ms

Παρόμοιο πλάνο με αυτό του ερωτήματος 13) και για τις δυο προσεγγίσεις έχουμε συναντήσει στο ερώτημα 5). Αρχικά γίνεται σειριακό σκανάρισμα στο πίνακα Customer και στη συνέχεια ο ίδιος πίνακας γίνεται Hash (έχει περιγραφεί σε προηγούμενο ερώτημα αυτή η διαδικασία). Παράλληλα γίνεται σειριακό σκανάρισμα στο πίνακα Reservation προχωρώντας στη πράξη Join των δύο πινάκων και εν τέλει στην εύρεση του αριθμού των κρατήσεων από την Ελλάδα.

14)Ακύρωσε όλες τις κρατήσεις που πρόκειται να αφιχθούν τον Ιανουάριο 2014.

```
EXPLAIN ANALYZE UPDATE Reservation
SET Cancelled=1
WHERE DATE_PART('month',Arrival)=1 AND DATE_PART('year',Arrival)=2014;
```

1η προσέγγιση

Update on reservation (cost=0.00..9706.00 rows=8 width=34) (actual time=90.338..90.338 rows=0 loops=1)
-> Seq Scan on reservation (cost=0.00..9706.00 rows=8 width=34) (actual time=90.335..90.335 rows=0 loops=1)
Filter: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))
Rows Removed by Filter: 300000

Startup Cost: 0.00
Total Cost: 9706.00
Plan Rows:8
Return Rows:0
Plan Width:34
Startup time: 90.338 ms
Max time: 90.338 ms
Total runtime: 90.382 ms

2η προσέγγιση

Update on reservation (cost=0.00..32353.00 rows=25 width=34) (actual time=334.087..334.087 rows=0 loops=1)

-> Seq Scan on reservation (cost=0.00..32353.00 rows=25 width=34) (actual time=334.082..334.082 rows=0 loops=1)

Filter: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))

Rows Removed by Filter: 1000000

Total runtime: 334.217 ms

10. Ορισμός ευρετηρίων για τους πίνακες της ΒΔ.

Τα ευρετήρια δημιουργήθηκαν με τη σειρά εκτέλεσης των ερωτημάτων.

```
CREATE INDEX hash_index_on_room_booked ON Room
USING hash(Booked) WHERE Booked=0;

CREATE INDEX tree_index_on_room_booked ON Room
USING btree(Booked) WHERE Booked=0;


CREATE INDEX tree_index_on_room_booked_roomtype ON Room
USING btree(RoomType,Booked);

CREATE INDEX tree_index_on_FK_customer_id ON Reservation
USING btree(CustomerID);

CREATE INDEX tree_index_on_FK_customer_id_eq_0 ON Reservation
USING btree(CustomerID) WHERE Cancelled=0;

CREATE INDEX tree_index_on_FK_customer_id_cancelled ON Reservation
USING btree(CustomerID,Cancelled);

CREATE INDEX tree_index_on_FK_customer_id_cancelled_eq0 ON Reservation
USING btree(CustomerID,Cancelled) WHERE Cancelled=0;

CLUSTER Reservation USING tree_index_on_FK_customer_id_cancelled;

CREATE INDEX tree_index_on_FK_customer_id_eq_1 ON Reservation
USING btree(CustomerID) WHERE Cancelled=1;

CREATE INDEX tree_index_on_FK_customer_id_cancelled_eq1 ON Reservation
USING btree(CustomerID,Cancelled) WHERE Cancelled=1;

CREATE INDEX tree_index_on_FK_room_id ON Reservation
USING btree(RoomID);

CREATE INDEX tree_index_on_room_id_roomname ON Room
USING btree(RoomID,RoomName);

CREATE INDEX tree_index_onFK_Reservid ON Payment
USING btree(ReservationID) WHERE Paid=1;

CREATE INDEX tree_index_onFK_Reservid_paid ON Payment
USING btree(ReservationID,Paid) WHERE Paid=1;
```

```
CREATE INDEX tree_index_on_FK_CustomerID ON Reservation
USING btree(CustomerID) WHERE Cancelled=1;

CREATE INDEX tree_index_on_FK_CustomerID_Cancelled ON Reservation
USING btree(CustomerID,Cancelled) WHERE Cancelled=1;

CREATE INDEX tree_index_on_Cancelled ON Reservation
USING btree(Cancelled) WHERE Cancelled=1;

CREATE INDEX tree_index_onFK_paid ON Payment
USING btree(Paid) WHERE Paid=1;

CREATE INDEX tree_index_on_arrival_cancelled ON Reservation
USING btree(Arrival,Cancelled);

CREATE INDEX tree_index_on_CustomerID_Where_cancel_eq_0 ON Reservation
USING btree(CustomerID) WHERE Arrival BETWEEN '01/01/2013' AND '31/01/2013' AND Cancelled=0;

CREATE INDEX tree_index_on_total ON Payment
USING btree(Total desc);

CREATE INDEX tree_index_on_date_part_cancel ON Reservation
USING btree(DATE_PART('month',Arrival),Cancelled) WHERE DATE_PART('month',Arrival)=1 AND
Cancelled=1;

CREATE INDEX tree_index_on_FK_sysadmin_id ON Reservation
USING btree(SysAdminID);

CREATE INDEX tree_index_on_date_part_year ON Reservation
USING btree(DATE_PART('year',Arrival) asc)
WHERE DATE_PART('year',Arrival)=2013 ;

CREATE INDEX hash_index_on_country ON Customer
USING hash(Country);

CREATE INDEX tree_index_on_date_part_month_year ON Reservation
USING btree(DATE_PART('month',Arrival),DATE_PART('year',Arrival));
```

11. Επανάληψη του βήματος 9 και συγκριτική αντιπαράθεση των μεθόδων υπολογισμού και των επιδόσεων για τις δύο περιπτώσεις.

Τα ερωτήματα που εκτελέστηκαν με τη χρήση ευρετηρίων στη παρούσα βάση δεδομένων ήταν για τους πίνακες με συνολικές εγγραφές ο κάθε ένας ξεχωριστα ώς εξής:

<u>1η προσέγγιση</u>
Πίνακας Administrator:1 εγγραφή
Πίνακας SystemAdmin:10 εγγραφές
Πίνακας Customer:300000 εγγραφές
Πίνακας Payment:300000 εγγραφές
Πίνακας Reservation:300000 εγγραφές
Πίνακας Room:1000 εγγραφές

2η προσέγγιση
Πίνακας Administrator:1 εγγραφή
Πίνακας SystemAdmin:10 εγγραφές
Πίνακας Customer:1000000 εγγραφές
Πίνακας Payment:1000000 εγγραφές
Πίνακας Reservation:1000000 εγγραφές
Πίνακας Room:1000 εγγραφές

1)Εμφάνισε όλες τις πληροφορίες των δωματίων που είναι διαθέσιμα.

```
EXPLAIN ANALYZE SELECT *
FROM Room
WHERE Booked=0 ;
```

1η προσέγγιση
Seq Scan on room (cost=0.00..19.50 rows=510 width=18) (actual time=0.008..0.147 rows=510 loops=1)
Filter: (booked = 0)

Startup Cost: 0.00
Total Cost: 19.50
Plan Rows: 510
Return Rows: 510
Plan Width:18
Startup Time: 0.008 ms
Max Time: 0.147 ms
Total runtime: 0.176 ms

Στο πίνακα Room ορίσαμε τα ευρετήρια hash_index_on_room_booked και tree_index_on_room_booked. Εκτελώντας το ερώτημα 1) από αυτά που μάθαμε στη θεωρία θα έπρεπε να χρησιμοποιήσει το hash index επειδή είναι κατάλληλο για επιλογές ισότητας. Ο Optimizer επιλέγει το πλάνο που περιγράψαμε χωρίς τη χρήση ευρετηρίων.

2η προσέγγιση
Δεν παίζει ρόλο να υπολογίσουμε το ερώτημα μιας και οι εγγραφές δωματίων έχουν παραμείνει ίδιες.

2)Εμφάνισε τον αριθμό των δίκλινων δωματίων που είναι διαθέσιμα(ομοίως για μονόκλινα, τρίκλινα,τετράκλινα).

```
EXPLAIN ANALYZE SELECT COUNT(Room.RoomName)
FROM Room
WHERE Booked=0 AND RoomType='2';
```

1η προσέγγιση
Aggregate (cost=14.97..14.98 rows=1 width=6) (actual time=0.258..0.259 rows=1 loops=1)
-> Bitmap Heap Scan on room (cost=5.64..14.63 rows=133 width=6) (actual time=0.159..0.210 rows=132 loops=1)
Recheck Cond: ((roomtype = '2'::bpchar) AND (booked = 0))
-> Bitmap Index Scan on ***tree_index_on_room_booked_roomtype*** *(cost=0.00..5.61 rows=133 width=0) (actual time=0.128..0.128 rows=132 loops=1)*
Index Cond: ((roomtype = '2'::bpchar) AND (booked = 0))

Startup Cost: 14.97
Total Cost: 14.98
Plan Rows:1
Return Rows:1
Plan Width:6
Startup Time: 0.258 ms
Max Time: *0.259* ms
Total runtime: 0.333 ms

2η προσέγγιση
Ομοίως με τη πρώτη προσέγγιση διότι δεν έχει αλλάξει το μέγεθος του πίνακα Room.

Στο ερώτημα 2) παρατηρούμε τη χρήση του ευρετηρίου ***tree_index_on_room_booked_roomtype*** που "φέρνει" τις εγγραφές χωριστά. Αυτή είναι μια ακριβή πράξη σε σχέση με το σειριακό σκανάρισμα ,όμως επειδή δεν θέλουμε όλα τα πεδία του πίνακα είναι φθηνή πράξη. Ο όρος " *Bitmap"* φροντίζει για την ταξινόμηση των όρων. Η διαδικάσία του Bitmap Heap Scan σημαίνει ότι έχει βρεθεί ένα σύνολο από εγγραφές που θα επιστραφούν , κάνοντας τη πράξη στο τέλος που θέλουμε δηλαδή να μετρήσει ποια είναι τα διαθέσιμα δίκλινα δωμάτια.

Σύγκριση περιπτώσεων
Συγκρίνοντας τις δυο περιπτώσεις είναι εμφανές πως με χρήση ευρετηρίων έχουμε χαμηλότερο κόστος σε σχέση με τη πρώτη περίπτωση.

3)Εμφάνισε τον αριθμό των πελατών που έκαναν κράτηση στο ξενοδοχείο χωρίς αυτούς που ακύρωσαν τη κράτησή τους.

```
EXPLAIN ANALYZE SELECT COUNT(Reservation.CustomerID)
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=0;
```

1η προσέγγιση
Aggregate (cost=15079.39..15079.40 rows=1 width=4) (actual time=142.024..142.024 rows=1 loops=1)
-> Merge Join (cost=1.30..14704.84 rows=149820 width=4) (actual time=0.183..129.922 rows=150475 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Only Scan using customer_id on customer (cost=0.42..7800.42 rows=300000 width=4) (actual time=0.044..37.623 rows=300000 loops=1)
Heap Fetches: 0
-> Index Only Scan using ***tree_index_on_fk_customer_id_cancelled_eq0 on reservation*** *(cost=0.42..4282.27 rows=149820 width=4) (actual time=0.125..24.134 rows=150475 loops=1)*
Index Cond: (cancelled = 0)
Heap Fetches: 0

Startup Cost: 15079.39
Total Cost: 15079.40
Plan Rows:1
Return Rows:1
Plan Width:4
Startup Time: 142.024 ms
Max Time: 142.024 ms
Total runtime: 142.103 ms

2η προσέγγιση

Aggregate (cost=179000.78..179000.79 rows=1 width=4) (actual time=892.202..892.202 rows=1 loops=1)
-> Merge Join (cost=1.23..177744.12 rows=502667 width=4) (actual time=0.082..845.717 rows=500530 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Only Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=4) (actual time=0.039..493.175 rows=1000000 loops=1)
Heap Fetches: 1000000
*-> Index Scan using **tree_index_on_fk_customer_id_eq_0 on reservation** (cost=0.42..17986.41 rows=502667 width=4) (actual time=0.034..116.463 rows=500530 loops=1)*

Startup Cost: 179000.78
Total Cost: 179000.79
Plan Rows:1
Return Rows:1
Plan Width:4
Startup Time: 892.202 ms
Max Time: 892.202 ms
Total runtime: 892.318 ms

Παρατηρούμε πως στο ερώτημα 3) και στις 2 προσεγγίσεις γίνεται η χρήση του ευρετηρίου ***tree_index_on_fk_customer_id_cancelled_eq0 on reservation .*** Έπειτα γίνεται σκανάρισμα στο ευρετήριο Customer και τέλος γίνεται η πράξη Join στους δυο πίνακες.

Σύγκριση περιπτώσεων

Συγκριτικά με τη πρώτη περίπτωση, η δεύτερη περίπτωση με τη χρήση ευρετηρίων είναι γρηγορότερη όπως επίσης έχει και χαμηλότερο κόστος.

4)Εμφάνισε το όνομα, το επώνυμο και το τηλέφωνο των πελατών που ακύρωσαν την κράτησή τους.

```
EXPLAIN ANALYZE SELECT FirstName,LastName,TelePhone
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Cancelled=1;
```

1η προσέγγιση

Merge Join (cost=3.30..52203.11 rows=150180 width=303) (actual time=0.598..794.485 rows=149525 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..45303.42 rows=300000 width=307) (actual time=0.006..661.283 rows=299996 loops=1)
*-> Index Only Scan using **tree_index_on_fk_customerid_cancelled on reservation** (cost=0.42..4275.66 rows=150180 width=4) (actual time=0.586..31.286 rows=149525 loops=1)*
Index Cond: (cancelled = 1)
Heap Fetches: 0

Startup Cost: 3.30
Total Cost: 52203.11
Plan Rows: 150180
Return Rows: 149525
Plan Width: 303
Startup Time: 0.598 ms
Max Time: 794.485 ms
Total runtime: 802.995 ms

2η προσέγγιση

Merge Join (cost=1.23..177522.78 rows=497333 width=303) (actual time=0.080..814.304 rows=499470 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=307) (actual time=0.024..453.606 rows=999999 loops=1)
-> Index Scan using tree_index_on_fk_customerid on reservation (cost=0.42..17831.75 rows=497333 width=4) (actual time=0.048..105.837 rows=499470 loops=1)

Startup Cost: 1.23
Total Cost: 177522.78
Plan Rows: 497333
Return Rows: 499470
Plan Width:303
Startup Time: 0.080 ms
Max Time: 814.304 ms
Total runtime: 836.116 ms

Στο ερώτημα 4) στις δυο προσεγγίσεις παραπάνω παρατηρείται η χρήση του ευρετηρίου ***tree_index_on_fk_customerid on reservation.*** Στη συνέχεια πραγματοποιείται σκανάρισμα στο πίνακα Customer και η πράξη Join.

Σύγκριση περιπτώσεων

Συγκριτικά με τη πρώτη περίπτωση, η δεύτερη περίπτωση με τη χρήση ευρετηρίων είναι γρηγορότερη όπως επίσης έχει και χαμηλότερο κόστος.

5)Εμφάνισε το όνομα κάθε δωματίου και τον αριθμό συχνότητας της κράτησης κάθε δωματίου με βάση τον αριθμό συχνότητας κράτησης.(Σημείωση:Για να είναι ένα δωμάτιο αρκετά επισκέψιμο θα πρέπει ο αριθμός συχνότητας επισκέψεως να είναι τουλάχιστον 30 επισκέψεις)

```
EXPLAIN ANALYZE SELECT RoomName,COUNT(Room.RoomName) AS visits
FROM Room
INNER JOIN Reservation
ON Room.RoomID=Reservation.RoomID
GROUP BY Room.RoomName
HAVING COUNT(Room.RoomName)>30;
```

1η προσέγγιση

HashAggregate (cost=11610.50..11623.00 rows=1000 width=6) (actual time=184.692..184.843 rows=1000 loops=1)
Filter: (count(room.roomname) > 30)
-> Hash Join (cost=29.50..9360.50 rows=300000 width=6) (actual time=0.595..112.965 rows=300000 loops=1)
Hash Cond: (reservation.roomid = room.roomid)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.007..27.016 rows=300000 loops=1)
-> Hash (cost=17.00..17.00 rows=1000 width=10) (actual time=0.516..0.516 rows=1000 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 43kB
-> Seq Scan on room (cost=0.00..17.00 rows=1000 width=10) (actual time=0.004..0.226 rows=1000 loops=1)

Startup Cost: 11610.50
Total Cost: 11623.00
Plan Rows: 1000
Return Rows:1000
Plan Width:6
Startup Time: 184.692 ms
Max Time: 184.843 ms
Total runtime: 184.997 ms

2η προσέγγιση

HashAggregate (cost=38632.50..38645.00 rows=1000 width=6) (actual time=617.298..617.445 rows=1000 loops=1)
Filter: (count(room.roomname) > 30)
-> Hash Join (cost=29.50..31132.50 rows=1000000 width=6) (actual time=0.543..379.743 rows=1000000 loops=1)
Hash Cond: (reservation.roomid = room.roomid)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.014..107.301 rows=1000000 loops=1)
-> Hash (cost=17.00..17.00 rows=1000 width=10) (actual time=0.496..0.496 rows=1000 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 43kB
-> Seq Scan on room (cost=0.00..17.00 rows=1000 width=10) (actual time=0.006..0.242 rows=1000 loops=1)

Startup Cost: 38632.50
Total Cost: 38645.00
Plan Rows: 1000
Return Rows:1000
Plan Width:6
Startup Time: 617.298 ms
Max Time: 617.445 ms
Total runtime: 617.602 ms

Σύγκριση περιπτώσεων

Παρόλο που έγινε ο ορισμός των ευρετηρίων tree_index_on_FK_room_id και tree_index_on_room_id_roomname δεν παρουσιάζεται διαφορά στο πλάνο που είδαμε χωρίς τη χρήση ευρετηρίων.

6)Εμφάνισε το όνομα , το επώνυμο και το τηλέφωνο των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν.

```
EXPLAIN ANALYZE SELECT FirstName,LastName,TelePhone
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
INNER JOIN Payment
ON Reservation.ReservationID=Payment.ReservationID
WHERE Cancelled =1 AND Paid=1;
```

1η προσέγγιση

Merge Join (cost=22974.79..70328.46 rows=74649 width=303) (actual time=213.017..440.515 rows=74530 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..45303.42 rows=300000 width=307) (actual time=0.012..138.819 rows=299996 loops=1)
-> Materialize (cost=22971.91..23345.15 rows=74649 width=4) (actual time=213.000..249.111 rows=74530 loops=1)
-> Sort (cost=22971.91..23158.53 rows=74649 width=4) (actual time=212.996..238.749 rows=74530

loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 1008kB
-> Hash Join (cost=6705.44..15907.88 rows=74649 width=4) (actual time=60.272..175.188 rows=74530 loops=1)
Hash Cond: (reservation.reservationid = payment.reservationid)
-> Index Scan using ***tree_index_on_fk_customerid on reservation*** *(cost=0.42..5385.30 rows=150180 width=8) (actual time=0.026..35.951 rows=149525 loops=1)*
-> Hash (cost=4258.02..4258.02 rows=149120 width=4) (actual time=60.106..60.106 rows=149323 loops=1)
Buckets: 4096 Batches: 8 Memory Usage: 668kB
-> Index Only Scan using ***tree_index_onfk_reservid_paid on payment*** *(cost=0.42..4258.02 rows=149120 width=4) (actual time=0.063..28.806 rows=149323 loops=1)*
Index Cond: (paid = 1)
Heap Fetches: 0

Startup Cost: 22974.79
Total Cost: 70328.46
Plan Rows: 74649
Return Rows: 74530
Plan Width: 303
Startup Time: 213.017 ms
Max Time: 440.515 ms
Total runtime: 444.521 ms

<u>2η προσέγγιση</u>
Merge Join (cost=81684.97..239512.29 rows=248766 width=303) (actual time=772.558..1525.205 rows=249383 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=307) (actual time=0.025..451.788 rows=999999 loops=1)
-> Materialize (cost=81684.52..82928.35 rows=248766 width=4) (actual time=772.526..901.617 rows=249383 loops=1)
-> Sort (cost=81684.52..82306.43 rows=248766 width=4) (actual time=772.522..867.483 rows=249383 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 3408kB
-> Hash Join (cost=25991.41..55987.57 rows=248766 width=4) (actual time=232.020..618.694 rows=249383 loops=1)
Hash Cond: (payment.reservationid = reservation.reservationid)
-> Seq Scan on payment (cost=0.00..17906.00 rows=500200 width=4) (actual time=0.031..143.960 rows=498774 loops=1)
Filter: (paid = 1)
Rows Removed by Filter: 501226
-> Hash (cost=17831.75..17831.75 rows=497333 width=8) (actual time=231.359..231.359 rows=499470 loops=1)
Buckets: 4096 Batches: 32 Memory Usage: 618kB
-> Index Scan ***using tree_index_on_fk_customerid on reservation*** *(cost=0.42..17831.75 rows=497333 width=8) (actual time=0.044..128.935 rows=499470 loops=1)*

Startup Cost: 81684.97
Total Cost: 239512.29
Plan Rows: 248766
Return Rows: 249383
Plan Width:303
Startup Time: 772.558 ms
Max Time: 1525.205 ms
Total runtime: 1537.800 ms

Στο ερώτημα 6) ακολουθείται παρόμοιο πλάνο με αυτο που είδαμε χωρίς τη χρήση ευρετηρίων με τη μόνη διαφορά ότι χρησιμοποιούνται τα ευρετήρια ***tree_index_on_fk_customerid on reservation*** & ***tree_index_onfk_reservid_paid on payment*** στη πρώτη προσέγγιση και μόνο το ***tree_index_on_fk_customerid on reservation*** στη 2η προσέγγιση.

Σύγκριση περιπτώσεων
Συγκρίνοντας τις δυο περιπτώσεις εκείνες δεν διαφέρουν πολύ , απλά με τη χρήση ευρετηρίων έχουμε μικρό κέρδος σε κόστος και καλύτερο χρόνο υπολογισμού.

7)Εμφάνισε όλες τις πληροφορίες των πελατών που είχαν προπληρώσει την κράτησή τους αλλά τελικά την ακύρωσαν ταξινομημένα ώς προς το συνολικό ποσό σε φθίνουσα σειρά.

```
EXPLAIN ANALYZE SELECT *
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
INNER JOIN Payment
ON Reservation.ReservationID=Payment.ReservationID
WHERE cancelled =1 AND paid=1
ORDER BY Total DESC;
```

1η προσέγγιση
Sort (cost=141830.75..142017.37 rows=74649 width=957) (actual time=739.918..782.382 rows=74530 loops=1)
Sort Key: payment.total
Sort Method: external merge Disk: 70976kB
-> Merge Join (cost=25666.04..73019.72 rows=74649 width=957) (actual time=207.048..453.210 rows=74530 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..45303.42 rows=300000 width=913) (actual time=0.010..151.007 rows=299996 loops=1)
-> Materialize (cost=25663.16..26036.41 rows=74649 width=44) (actual time=207.031..232.333 rows=74530 loops=1)
-> Sort (cost=25663.16..25849.79 rows=74649 width=44) (actual time=207.029..221.673 rows=74530 loops=1)
Sort Key: reservation.customerid
Sort Method: external sort Disk: 4224kB
-> Merge Join (cost=1.39..17321.64 rows=74649 width=44) (actual time=0.056..156.136 rows=74530 loops=1)
Merge Cond: (reservation.reservationid = payment.reservationid)
-> Index Scan using reservation_id on reservation (cost=0.42..10759.42 rows=150180 width=30) (actual time=0.012..74.963 rows=149525 loops=1)
Filter: (cancelled = 1)
Rows Removed by Filter: 150475
-> Index Scan using ***tree_index_onfk_reservid on payment*** *(cost=0.42..5068.02 rows=149120 width=14) (actual time=0.039..31.211 rows=149320 loops=1)*

Startup Cost: 141830.75
Total Cost: 142017.37
Plan Rows: 74649
Return Rows: 74530
Plan Width:957
Startup Time: 739.918 ms
Max Time: 782.382 ms
Total runtime: 799.961 ms

2η προσέγγιση

Sort (cost=593868.23..594490.15 rows=248766 width=957) (actual time=2972.492..3118.741 rows=249383 loops=1)
Sort Key: payment.total
Sort Method: external merge Disk: 237456kB
-> Merge Join (cost=99984.97..257812.29 rows=248766 width=957) (actual time=885.302..1785.303 rows=249383 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=913) (actual time=0.010..511.505 rows=999999 loops=1)
-> Materialize (cost=99984.52..101228.35 rows=248766 width=44) (actual time=885.285..1039.603 rows=249383 loops=1)
-> Sort (cost=99984.52..100606.43 rows=248766 width=44) (actual time=885.282..993.928 rows=249383 loops=1)
Sort Key: reservation.customerid
Sort Method: external merge Disk: 14120kB
-> Hash Join (cost=27448.41..62380.57 rows=248766 width=44) (actual time=215.313..661.275 rows=249383 loops=1)
Hash Cond: (payment.reservationid = reservation.reservationid)
-> Seq Scan on payment (cost=0.00..17906.00 rows=500200 width=14) (actual time=0.021..138.351 rows=498774 loops=1)
Filter: (paid = 1)
Rows Removed by Filter: 501226
-> Hash (cost=17831.75..17831.75 rows=497333 width=30) (actual time=215.096..215.096 rows=499470 loops=1)
Buckets: 2048 Batches: 32 Memory Usage: 992kB
*-> Index Scan using **tree_index_on_fk_customerid on reservation** (cost=0.42..17831.75 rows=497333 width=30) (actual time=0.025..112.771 rows=499470 loops=1)*

Startup Cost: 593868.23
Total Cost: 594490.15
Plan Rows: 248766
Return Rows: 249383
Plan Width: 957
Startup Time: 2972.492 ms
Max Time: 3118.741 ms
Total runtime: 3113.933 ms

Με παρόμοιο τρόπο και στο ερώτημα 7) ακολουθείται παρόμοιο πλάνο με τη μόνη διαφορά ότι χρησιμοποιείται το ευρετήριο ***tree_index_on_fk_customerid on reservation*** και συγκριτικά οι δυο περιπτώσεις έχουν σχεδόν τα ίδια κόστη και χρόνους υπολογισμού με λίγο καλύτερη περίπτωση εκείνη με τη χρήση ευρετηρίων.

8)Εμφάνισε το όνομα , το επώνυμο και την ημερομηνία που οι πελάτες ήρθαν στο ξενοδοχείο κατά το μήνα Ιανουάριο(Δεν ακύρωσαν τη κράτησή τους).

```
EXPLAIN ANALYZE SELECT Customer.FirstName,Customer.LastName , Reservation.Arrival
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE Arrival BETWEEN '01/01/2013' AND '31/01/2013' AND Cancelled=0;
```

1η προσέγγιση

Nested Loop (cost=0.70..30253.36 rows=4497 width=206) (actual time=0.076..37.038 rows=4813 loops=1)
*-> Index Scan using **tree_index_on_customerid_where_cancel_eq_0** on reservation (cost=0.28..198.47 rows=4497 width=8) (actual time=0.064..5.908 rows=4813 loops=1)*
-> Index Scan using customer_id on customer (cost=0.42..6.67 rows=1 width=206) (actual time=0.005..0.006 rows=1 loops=4813)
Index Cond: (customerid = reservation.customerid)

Startup Cost: 0.70
Total Cost: 30253.36
Plan Rows: 4497
Return Rows: 4813
Plan Width: 206
Startup Time: 0.076 ms
Max Time: 37.038 ms
Total runtime: 37.536 ms

Στη πρώτη προσέγγιση γίνεται σκανάρισμα στο ευρετήριο ***tree_index_on_customerid_where_cancel_eq_0*** και στη συνέχεια στο πίνακα Customer ικανοποιώντας τη συνθήκη ευρετηρίου customerid = reservation.customerid . Ολόκληρη η διαδικασία που περιγράψαμε γίνεται σε ένα Nested Loop που σήμαίνει ότι θα γίνει η πράξη Join με γνώμονα τα αποτελέσματα που θα λαμβάνει με καποια κριτήρια που δεν είναι άλλα από τις συνθήκες που έχουμε θέσει.

2η προσέγγιση

Merge Join (cost=0.76..154966.29 rows=27761 width=206) (actual time=0.127..248.640 rows=31093 loops=1)
Merge Cond: (customer.customerid = reservation.customerid)
-> Index Scan using customer_id on customer (cost=0.42..150983.42 rows=1000000 width=206) (actual time=0.025..188.586 rows=400000 loops=1)
-> Index Scan using ***tree_index_on_customerid_where_cancel_eq_0 on reservation*** (cost=0.29..1144.91 rows=27761 width=8) (actual time=0.037..11.910 rows=31093 loops=1)

Startup Cost: 0.76
Total Cost: 154966.29
Plan Rows: 27761
Return Rows: 31093
Plan Width: 206
Startup Time: 0.127 ms
Max Time: 248.640 ms
Total runtime: 250.301 ms

Στη δεύτερη προσέγγιση ακολουθείται διαφορετικό πλάνο σε σχέση με τη πρώτη προσέγγιση. Έτσι πραγματοποιείται σκανάρισμα στο πίνακα Customer και σκανάρισμα στο ευρετήριο ***tree_index_on_customerid_where_cancel_eq_0*** συνεχίζοντας με τη πράξη του Join και συγκεκριμένα με Merge Join.

Σύγκριση περιπτώσεων

Συγκρίνοντας τις δύο περιπτώσεις είναι εύκολα προφανές πως με τη χρήση ευρετηρίων ελαχιστοποιούμε το κόστος αλλά και το χρόνο υπολογισμού μιας και ο optimizer επιλέγει το καλύτερο πλάνο με βάσει τα ευρετήρια που ορίσαμε. Στο συγκεκριμένο ερώτημα παρατηρούμε τη χρήση του B+ δέντρου το οποίο είναι κατάλληλο για ερωτήματα διαστήματος μιας και κάνουμε χρήση ενός τέτοιου είδους ερωτήματος.

9)Εμφάνισε όλες τις πληροφορίες του πελάτη που έδωσε στο ξενοδοχείο το μεγαλύτερο ποσό κατά τη διαμονή του.

```
EXPLAIN ANALYZE SELECT *
FROM Customer
WHERE Customer.CustomerID=(SELECT Reservation.CustomerID
FROM Reservation
WHERE Cancelled=0 AND Reservation.ReservationID=(SELECT Payment.ReservationID
FROM Payment
WHERE Total=(SELECT MAX(Total)
FROM Payment)));
```

1η προσέγγιση

Index Scan using customer_id on customer (cost=17.77..25.78 rows=1 width=913) (actual time=0.178..0.179 rows=1 loops=1)
Index Cond: (customerid = $3)
InitPlan 4 (returns $3)
-> Index Scan using reservation_id on reservation (cost=9.32..17.34 rows=1 width=4) (actual time=0.159..0.160 rows=1 loops=1)
Index Cond: (reservationid = $2)
Filter: (cancelled = 0)
InitPlan 3 (returns $2)
*-> Index Scan using **tree_index_on_total** on payment payment_1 (cost=0.88..8.90 rows=1 width=4) (actual time=0.112..0.114 rows=1 loops=1)*
Index Cond: (total = $1)
InitPlan 2 (returns $1)
-> Result (cost=0.45..0.46 rows=1 width=0) (actual time=0.086..0.087 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Limit (cost=0.42..0.45 rows=1 width=4) (actual time=0.082..0.083 rows=1 loops=1)
*-> Index Only Scan using **tree_index_on_total** on payment (cost=0.42..8550.42 rows=300000 width=4) (actual time=0.079..0.079 rows=1 loops=1)*
Index Cond: (total IS NOT NULL)
Heap Fetches: 0

Startup Cost: 17.77
Total Cost: 25.78
Plan Rows:1
Return Rows:1
Plan Width: 913
Startup Time: 0.178 ms
Max Time: 0.179 ms
Total runtime: 0.288 ms

2η προσέγγιση

Index Scan using customer_id on customer (cost=21.82..29.83 rows=1 width=913) (actual time=0.161..0.162 rows=1 loops=1)
Index Cond: (customerid = $3)
InitPlan 4 (returns $3)
-> Index Scan using reservation_id on reservation (cost=13.37..21.39 rows=1 width=4) (actual time=0.145..0.145 rows=1 loops=1)
Index Cond: (reservationid = $2)
Filter: (cancelled = 0)
InitPlan 3 (returns $2)
*-> Index Scan using **tree_index_on_total** on payment payment_1 (cost=0.91..12.95 rows=2 width=4) (actual time=0.128..0.129 rows=1 loops=1)*
Index Cond: (total = $1)
InitPlan 2 (returns $1)
-> Result (cost=0.48..0.49 rows=1 width=0) (actual time=0.122..0.122 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Limit (cost=0.42..0.48 rows=1 width=4) (actual time=0.119..0.120 rows=1 loops=1)
*-> Index Only Scan using **tree_index_on_total** on payment (cost=0.42..50104.36 rows=1000000 width=4) (actual time=0.117..0.117 rows=1 loops=1)*
Index Cond: (total IS NOT NULL)
Heap Fetches: 1

Startup Cost: 21.82
Total Cost: 29.83
Plan Rows:1
Return Rows:1
Plan Width: 913
Startup Time: 0.161 ms
Max Time: 0.162 ms
Total runtime: 0.288 ms

Στις δυο προσεγγίσεις έχουμε να αντιμετωπίσουμε από ένα query με sub-selects . Αρχικά γίνεται σκανάρισμα στο ευρετήριο ***tree_index_on_total*** . Έπειτα επιστρέφεται στο InitPlan 1 το MAX(Total) που χρησιμοποιήται ως εισοδος στο αμέσως προηγούμενο sub-select χρησιμοποιωντας το ευρετήριο ***tree_index_on_total*** επιστρέφοντας το ReservationID από το πίνακα Payment. Στη συνέχεια σκανάρεται ο πίνακας Reservation και γίνεται σύγκριση το πρωτεύον κλειδί του με το αποτέλεσμα που βρίκαμε προηγουμένως και επιστρεφει το CustomerID βρίσκοντας τέλος τις πληροφορίες του πελάτη που έδωσε το μεγαλύτερο ποσό κατά τη διαμονή του.

Σύγκριση περιπτώσεων

Αυτή η περίπτωση χρήσης ευρετηρίων είναι η πιο αποδοτική από αυτές που είδαμε έως τώρα. Οι διαφορές με τη χρήση σε σχέση με τα αποτελέσματα χωρίς τη χρήση ευρετηρίων είναι βελτιστοποιημένες και σε κόστος αλλά κυριως σε χρόνο.

10)Εμφάνισε το όνομα και το επώνυμο των πελατών που ακύρωσαν την άφιξή τους στο ξενοδοχείο κατά το μήνα Ιανουάριο.

```
EXPLAIN ANALYZE SELECT Customer.FirstName,Customer.LastName
FROM Customer
INNER JOIN Reservation
ON Customer.CustomerID=Reservation.CustomerID
WHERE DATE_PART('month',Arrival)=1 AND Cancelled=1;
```

1η προσέγγιση

Nested Loop (cost=0.71..6603.70 rows=751 width=202) (actual time=0.045..31.704 rows=4739 loops=1)
-> Index Scan using ***tree_index_on_date_part_cancel on reservation*** *(cost=0.29..1159.87 rows=751 width=4) (actual time=0.036..3.226 rows=4739 loops=1)*
Index Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (cancelled = 1))
-> Index Scan using customer_id on customer (cost=0.42..7.24 rows=1 width=206) (actual time=0.005..0.005 rows=1 loops=4739)
Index Cond: (customerid = reservation.customerid)

Startup Cost: 0.71
Total Cost: 6603.70
Plan Rows: 751
Return Rows: 4739
Plan Width: 202
Startup Time: 0.045 ms
Max Time: 31.704 ms
Total runtime: 32.135 ms

Παρόμοια περίπτωση πλάνου έχουμε ξαναδεί και σε προηγούμενο ερώτημα. Πιο συγκεκριμένα γίνεται σκανάρισμα στο ευρετήριο ***tree_index_on_date_part_cancel*** του πίνακα Reservation με βάση τις συνθήκες που βρίσκονται στο where και στο πίνακα Customer ομοίως βλέποντας εαν το *customerid = reservation.customerid.* Η παραπάνω διαδικασία εκτελείται σε ένα Nested Loop.

2η προσέγγιση

Nested Loop (cost=66.34..25004.87 rows=2487 width=202) (actual time=19.431..290.181 rows=80966 loops=1)
-> Bitmap Heap Scan on reservation (cost=65.91..5193.29 rows=2487 width=4) (actual time=19.376..40.273 rows=80966 loops=1)
Recheck Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (cancelled = 1))
-> Bitmap Index Scan on ***tree_index_on_date_part_cancel*** *(cost=0.00..65.29 rows=2487 width=0) (actual time=18.457..18.457 rows=80966 loops=1)*
Index Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (cancelled = 1))
-> Index Scan using customer_id on customer (cost=0.42..7.96 rows=1 width=206) (actual time=0.002..0.003 rows=1 loops=80966)
Index Cond: (customerid = reservation.customerid)

Startup Cost: 66.34
Total Cost: 25004.87
Plan Rows: 2487
Return Rows: 80966
Plan Width: 202
Startup Time: 19.431 ms
Max Time: 290.181 ms
Total runtime: 295.320 ms

Στη 2η προσέγγιση γίνεται σκανάρισμα στο σωρό στο πίνακα Reservation ελέγχοντας τις συνθήκες που βρίσκονται στο Where . Στη συνέχεια γίνεται σκανάρισμα στο ευρετήριο ***tree_index_on_date_part_cancel*** και τέλος γίνεται σκανάρισμα στο πίνακα Customer βλέποντας εαν το *customerid = reservation.customerid.*Η παραπάνω διαδικασία εκτελείται σε ένα Nested Loop.

Σύγκριση περιπτώσεων

Παρατηρώντας τα αποτελέσματα από τη χρήση ευρετηρίων και συγκρίνοντάς τα με εκείνα που είδαμε πριν τη χρήση ευρετηρίων παρατηρούμε πως η χρήση ευρετηρίων βοηθά τον optimizer να επιλέξει το καλύτερο δυνατό πλάνο με αποτέλεσμα τη μείωση του κόστους εκτέλεσης του ερωτήματος ,όπως επίσης και σε χαμηλότερο χρόνο σε σχέση με πριν.

11)Εμφάνισε το όνομα του δωματίου που οι πελάτες έκαναν τις περισσότερες κρατήσεις.

```
EXPLAIN ANALYZE SELECT Room.RoomName
FROM Room
WHERE Room.RoomID=(
SELECT T.RoomID
FROM(SELECT  Reservation.RoomID,COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) AS T
WHERE K=(SELECT MAX(K)
FROM(SELECT COUNT(*) AS K
FROM Reservation
GROUP BY Reservation.RoomID) l)
GROUP BY T.RoomID);
```

1η προσέγγιση

*Index Only Scan using **tree_index_on_room_id_roomname on room** (cost=14211.78..14215.80 rows=1 width=6) (actual time=181.389..181.390 rows=1 loops=1)*
Index Cond: (roomid = $1)
Heap Fetches: 0
InitPlan 2 (returns $1)
-> HashAggregate (cost=14209.51..14211.51 rows=200 width=4) (actual time=181.342..181.342 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Aggregate (cost=6728.50..6728.51 rows=1 width=8) (actual time=87.521..87.521 rows=1 loops=1)
-> HashAggregate (cost=6706.00..6716.00 rows=1000 width=4) (actual time=87.309..87.425 rows=1000 loops=1)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.002..21.802 rows=300000 loops=1)
-> HashAggregate (cost=7456.00..7468.50 rows=1000 width=4) (actual time=181.291..181.338 rows=1 loops=1)
Filter: (count(*) = $0)
Rows Removed by Filter: 999
-> Seq Scan on reservation reservation_1 (cost=0.00..5206.00 rows=300000 width=4) (actual time=0.011..25.140 rows=300000 loops=1)

Startup Cost: 14211.78
Total Cost: 14215.80
Plan Rows:1
Return Rows:1
Plan Width:6
Startup Time: 181.389 ms
Max Time: 181.390 ms
Total runtime: 181.541 ms

Παρόμοιο πλάνο με αυτό που είδαμε στη μελέτη της πρώτης προσέγγισης του ερωτήματος 11) με τη διαφορά ότι επιλέγει το ευρετήριο ***tree_index_on_room_id_roomname on room*** με πολύ μικρή διαφορά σε κόστος και χρόνο υπολογισμού**.**

2η προσέγγιση

*Index Only Scan using **tree_index_on_room_id_roomname on room** (cost=47255.78..47263.80 rows=1 width=6) (actual time=634.864..634.865 rows=1 loops=1)*
Index Cond: (roomid = $1)
Heap Fetches: 1
InitPlan 2 (returns $1)
-> HashAggregate (cost=47253.51..47255.51 rows=200 width=4) (actual time=634.810..634.811 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Aggregate (cost=22375.50..22375.51 rows=1 width=8) (actual time=300.615..300.615 rows=1 loops=1)
-> HashAggregate (cost=22353.00..22363.00 rows=1000 width=4) (actual time=300.421..300.539 rows=1000 loops=1)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.006..89.171 rows=1000000 loops=1)
-> HashAggregate (cost=24853.00..24865.50 rows=1000 width=4) (actual time=634.801..634.805 rows=1 loops=1)
Filter: (count(*) = $0)
Rows Removed by Filter: 999
-> Seq Scan on reservation reservation_1 (cost=0.00..17353.00 rows=1000000 width=4) (actual time=0.011..100.361 rows=1000000 loops=1)

Startup Cost: 47255.78
Total Cost: 47263.80
Plan Rows:1
Return Rows:1
Plan Width:6
Startup Time: 634.864 ms
Max Time: 634.865 ms
Total runtime: 634.956 ms

Ομοίως με τη πρώτη προσέγγιση.

Σύγκριση περιπτώσεων

Οι δυο προσεγγίσεις παραπάνω σε σχέση με αυτές που ορίσαμε χωρίς τη χρήση ευρετηρίων δεν παρουσιάζουν ουσιαστική διαφορά σε κόστος και χρόνο.

12)Εμφάνισε το συνολικό αριθμό αφίξεων,για κάθε μήνα,που έγιναν το έτος 2013 σε αύξουσα σειρά με βάση τον μήνα.

```
EXPLAIN ANALYZE SELECT DATE_PART('month',Arrival),COUNT(ReservationID) AS
NumOfReservations
FROM Reservation
WHERE DATE_PART('year',Arrival)=2013
GROUP BY DATE_PART('month',Arrival)
ORDER BY DATE_PART('month',Arrival);
```

1η προσέγγιση

Sort (cost=76.72..76.72 rows=2 width=8) (actual time=204.780..204.782 rows=12 loops=1)
Sort Key: (date_part('month'::text, (arrival)::timestamp without time zone))
Sort Method: quicksort Memory: 25kB
-> HashAggregate (cost=76.68..76.71 rows=2 width=8) (actual time=204.758..204.760 rows=12 loops=1)
*-> Index Scan using **tree_index_on_date_part_year on reservation** (cost=0.43..69.18 rows=1500 width=8) (actual time=0.039..135.469 rows=300000 loops=1)*
Index Cond: (date_part('year'::text, (arrival)::timestamp without time zone) = 2013::double precision)

Startup Cost: 76.72
Total Cost: 76.72
Plan Rows:2
Return Rows:12
Plan Width:8
Startup Time: 204.780 ms
Max Time: 204.782 ms
Total runtime: 204.815 ms

Στη πρώτη προσέγγιση γίνεται σκανάρισμα στο ευρετήριο ***tree_index_on_date_part_year*** συνεχίζοντας με τη πράξη *HashAggregate* μιας και είναι πολύ πιο γρήγορη από το να κάνει Sort & Group. Τέλος γίνεται η ταξινόμηση με χρήση της Quick Sort.

2η προσέγγιση

Sort (cost=7184.45..7184.46 rows=3 width=8) (actual time=365.995..365.996 rows=12 loops=1)
Sort Key: (date_part('month'::text, (arrival)::timestamp without time zone))
Sort Method: quicksort Memory: 25kB
-> HashAggregate (cost=7184.38..7184.43 rows=3 width=8) (actual time=365.939..365.940 rows=12 loops=1)
-> Bitmap Heap Scan on reservation (cost=95.18..7159.38 rows=5000 width=8) (actual time=66.616..242.239 rows=600000 loops=1)
Recheck Cond: (date_part('year'::text, (arrival)::timestamp without time zone) = 2013::double precision)
-> Bitmap Index Scan on ***tree_index_on_date_part_year*** *(cost=0.00..93.93 rows=5000 width=0) (actual time=65.924..65.924 rows=600000 loops=1)*
Index Cond: (date_part('year'::text, (arrival)::timestamp without time zone) = 2013::double precision)

Startup Cost: 7184.45
Total Cost: 7184.46
Plan Rows:3
Return Rows:12
Plan Width:8
Startup Time: 365.995 ms
Max Time: 365.996 ms
Total runtime: 366.084 ms

Η δεύτερη προσέγγιση έχει ομοιότητα με τη πρώτη με τη διαφορά ότι παρεμβαίνει η πράξη *Bitmap Heap Scan* στο πίνακα Reservation που σημαίνει ότι βρήκε ένα υποσύνολο εγγραφών το οποίο θα επιστρέψει, πράγμα που είναι αρκετά γρήγορο.

Σύγκριση περιπτώσεων

Ακόμα ένα αποτέλεσμα που δείχνει τα αποτελέσματα στη μείωση κόστους και χρόνου υπολογισμού που έχει η χρήση ευρετηρίων σε πίνακες όταν θέλουμε να εκτελέσουμε αποδοτικά και βελτιστοποιημένα ερωτήματα.

13) Εμφάνισε τον αριθμό των κρατήσεων που έγιναν από την Ελλάδα.

```
EXPLAIN ANALYZE SELECT COUNT(ReservationID)
FROM Reservation
INNER JOIN Customer
ON Customer.CustomerID=Reservation.CustomerID
WHERE Country='Greece';
```

1η προσέγγιση

Aggregate (cost=10521.49..10521.50 rows=1 width=4) (actual time=74.546..74.547 rows=1 loops=1)
-> Hash Join (cost=4175.45..10518.48 rows=1203 width=4) (actual time=7.200..74.418 rows=1159 loops=1)
Hash Cond: (reservation.customerid = customer.customerid)
-> Seq Scan on reservation (cost=0.00..5206.00 rows=300000 width=8) (actual time=0.006..32.723 rows=300000 loops=1)
-> Hash (cost=4160.41..4160.41 rows=1203 width=4) (actual time=7.145..7.145 rows=1159 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 41kB
-> Bitmap Heap Scan on customer (cost=37.32..4160.41 rows=1203 width=4) (actual time=0.391..6.750 rows=1159 loops=1)
Recheck Cond: (country = 'Greece'::bpchar)
*-> Bitmap Index Scan on **hash_index_on_country** (cost=0.00..37.02 rows=1203 width=0) (actual time=0.204..0.204 rows=1159 loops=1)*
Index Cond: (country = 'Greece'::bpchar)

Startup Cost: 10521.49
Total Cost: 10521.50
Plan Rows:1
Return Rows:1
Plan Width:4
Startup Time: 74.546 ms
Max Time: 74.547 ms
Total runtime: 74.610 ms

2η προσέγγιση

Aggregate (cost=38841.82..38841.83 rows=1 width=4) (actual time=265.452..265.452 rows=1 loops=1)
-> Hash Join (cost=13938.62..38831.78 rows=4016 width=4) (actual time=27.015..264.986 rows=3924 loops=1)
Hash Cond: (reservation.customerid = customer.customerid)
-> Seq Scan on reservation (cost=0.00..17353.00 rows=1000000 width=8) (actual time=0.014..91.020 rows=1000000 loops=1)
-> Hash (cost=13888.42..13888.42 rows=4016 width=4) (actual time=26.910..26.910 rows=3924 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 138kB
-> Bitmap Heap Scan on customer (cost=135.12..13888.42 rows=4016 width=4) (actual time=4.137..25.478 rows=3924 loops=1)
Recheck Cond: (country = 'Greece'::bpchar)
*-> Bitmap Index Scan on **hash_index_on_country** (cost=0.00..134.12 rows=4016 width=0) (actual time=2.309..2.309 rows=3924 loops=1)*
Index Cond: (country = 'Greece'::bpchar)

Startup Cost: 38841.82
Total Cost: 38841.83
Plan Rows:1
Return Rows:1
Plan Width:4
Startup Time: 265.452 ms
Max Time: 265.452 ms
Total runtime: 265.625 ms

Στο ερώτημα 13) , αρχικά ακολουθείται ένα πλάνο δυο βημάτων,δηλαδή γίνεται σκανάρισμα στο ευρετήριο ώστε να βρεθούν οι εγγραφές που ικανοποιούν τη συνθήκη μας (Country='Greece';) και το επόμενο βήμα είναι να φέρει ως αποτέλεσμα αυτές τις γραμμές μόνο (Heap). Στη συνέχει γίνεται η πράξη Hash. Παράλληλα έχει γίνει σειριακό σκανάρισμα στο πίνακα Reservation.Τέλος γίνεται η πράξη Join και συγκεκριμένα Hash Join καθώς και το μέτρημα των κρατήσεων από την Ελλάδα.

Σύγκριση περιπτώσεων

Παρατηρούμε πως τη χρήση ευρετηρίου ***hash_index_on_country*** έχουμε μικρότερα κόστη καθώς και καλύτερους χρόνους σε σχέση με τη περίπτωση χωρίς τη χρήση ευρετηρίων και φαίνεται και το πόσο αποδοατικό είναι το να ορίσουμε ευρετήριο Hash που είναι κατάλληλο για ελέγχους ισότητας .

14)Ακύρωσε όλες τις κρατήσεις που πρόκειται να αφιχθούν τον Ιανουάριο 2014.

```
EXPLAIN ANALYZE UPDATE Reservation
SET Cancelled=1
WHERE DATE_PART('month',Arrival)=1 AND DATE_PART('year',Arrival)=2014;
```

ΠΡΟΣΟΧΗ!!!

Για τη λήψη αποτελεσμάτων θα πρέπει να εισάγετε το τελευταίο ευρετήριο επειδή δεν το εχω εισάγει στα backup αρχεία.

```
CREATE INDEX tree_index_on_date_part_month_year ON Reservation
USING btree(DATE_PART('month',Arrival),DATE_PART('year',Arrival));
```

Επειδή γίνεται ενημέρωση θα επηρεάζονταν προηγούμενα ερωτήματα.

1η προσέγγιση

Update on reservation (cost=4.51..35.27 rows=8 width=34) (actual time=0.079..0.079 rows=0 loops=1)
-> Bitmap Heap Scan on reservation (cost=4.51..35.27 rows=8 width=34) (actual time=0.076..0.076 rows=0 loops=1)
Recheck Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))
-> Bitmap Index Scan on ***tree_index_on_date_part_month_year*** *(cost=0.00..4.51 rows=8 width=0) (actual time=0.069..0.069 rows=0 loops=1)*
Index Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))

Startup Cost: 4.51
Total Cost: 35.27
Plan Rows:8
Return Rows:0
Plan Width:34
Startup Time: 0.079 ms
Max Time: 0.079 ms
Total runtime: 0.305 ms

2η προσέγγιση

Update on reservation (cost=4.69..100.94 rows=25 width=34) (actual time=0.082..0.082 rows=0 loops=1)
-> Bitmap Heap Scan on reservation (cost=4.69..100.94 rows=25 width=34) (actual time=0.080..0.080 rows=0 loops=1)
Recheck Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))
-> Bitmap Index Scan on ***tree_index_on_date_part_month_year*** *(cost=0.00..4.68 rows=25 width=0) (actual time=0.075..0.075 rows=0 loops=1)*
Index Cond: ((date_part('month'::text, (arrival)::timestamp without time zone) = 1::double precision) AND (date_part('year'::text, (arrival)::timestamp without time zone) = 2014::double precision))

Startup Cost: 4.69
Total Cost: 100.94
Plan Rows:25
Return Rows:0
Plan Width:34
Startup Time: 0.082 ms
Max Time: 0.082 ms
Total runtime: 0.230 ms

Στο ερώτημα 14) ακολουθείται παρόμοιο πλάνο που έχουμε δει σε προηγούμενο ερώτημα για την ενημέρωση των εγγραφών.

Σύγκριση περιπτώσεων
Η βασική διαφορά που παρατηρούμε είναι η χρήση ευρετηρίου ***tree_index_on_date_part_month_year*** έχει κόστος ενώ χωρίς τη χρήση ευρετηρίων το κόστος είναι μηδενικό. Χωρίς τη χρήση ευρετηρίου ό χρόνος είναι μεγαλύτερος σε σχέση με τη χρήση ευρετηρίου.

12. Συμπεράσματα

Ολοκληρώνοντας τη παρούσα πρακτική εργασία εξάγουμε ορισμένα συμπεράσματα αλλά και τη σημασία χρήσης ευρετηρίων στην επιλογή του Optimizer για το πιο αποδοτικό πλάνο. Πιο συγκεκριμένα παρατήρησα:
- Την αυτόματη δημιουργία ευρετηρίων για τα πεδία της βάσης που ήταν ορισμένα είτε ως πρωτεύοντα κλειδιά είτε με την ιδιότητα της μοναδικότητας (UNIQUE) στη περίπτωση μελέτης ερωτημάτων χωρίς τον ορισμό ευρετηρίων. Αυτό γίνεται για να υπάρχει αποδοτική πρόσβαση στα διάφορα ερωτήματα που θα εκτελέσουμε.
- Μπορούμε να χρησιμοποιήσουμε μόνο ένα CLUSTERED Index σε ένα πίνακα λόγω του ότι τα δεδομένα μπορούν να είναι ταξινομημένα μόνο σε μία διάταξη.
- Επίσης παρατήρησα πως η PostgreSQL αξιοποιεί αποδοτικά τα B+ δέντρα μιας και μπορούμε να ορίσουμε ευρετήρια με σύνθετα κλειδιά αναζήτησης σε αντίθεση με τα Hash indexes που μας επιτρέπουν μόνο ένα κλειδί αναζήτησης.
- Τα B+ δέντρα είναι καλά για ερωτήματα διαστήματος αλλά και ελέγχους ισότητας.
- Τα Hash indexes είναι καλύτερα σε ελέγχους ισότητας από τα B+ δέντρα.
- Ακόμα έγινε ορισμός μερικών ευρετηρίων (Partial Indexes) με τη λογική πως εάν έχουμε 1000000 εγγραφές σε ένα πίνακα και θέλουμε να αποκλείσουμε κάποιες από αυτές είναι καλή επιλογή να δημιουργήσουμε μερικά ευρετήρια λόγω της εξοικονόμησης δίσκου αλλά και για να κάνουμε περισσότερο αποτελεσματική τη σάρωση του ευρετηρίου. Ο κύριος περιοριστικός παράγοντας που έχουν τα μερικά ευρετήρια είναι πως πρέπει να χρησιμοποιήσουμε την ίδια συνθήκη στο WHERE του ερωτήματος μας με αυτή που ορίσαμε στο ευρετήριο ώστε να γίνει αποδοτική η χρήση του.
- Η καλύτερη πρακτική για να βελτιστοποιήσουμε ένα ερώτημα που θέλουμε να εκτελέσουμε είναι να το ορίσουμε με τέτοιο τρόπο ώστε να είναι κατάλληλο με βάση τη συνθήκη που έχουμε ορίσει στο where του ερωτήματος μας όπως επίσης και άλλες πρακτικές.
- Επιπροσθέτως παρατήρηση τη δυσκολία του να βελτιστοποιήσουμε τη σύζευξη τριών πινάκων μιας και οι διαφορές στα αποτελέσματα με και χωρίς τη χρήση ευρετηρίων ήταν οριακά καλύτερα με τη χρήση ευρετηρίων. Η χρήση ευρετηρίων ήταν πολύ αποδοτική για σύζευξη δύο πινάκων.
- Τέλος το βασικό συμπέρασμα που βγάζουμε από τη παρούσα εργασία είναι η αναγκαιότητα χρήσης ευρετηρίων για βάσεις δεδομένων με πολύ μεγάλο όγκο δεδομένων.

13 . Λίστα παραδοτέων αρχείων

1) Η διαδικασία για την εγκατάσταση της βάσης δεδομένων βρίσκεται στο αρχείο ***pgadmin_guide_for_db.pdf*** .

2) Το διάγραμμα οντοτήτων – συσχετίσεων βρίσκεται στο αρχείο ***E_R_Hotel.jpg*** (επισυνάπτεται και το αρχείο ***E_R_Hotel.dia***).

3) Το σχεσιακό σχήμα βρίσκεται στο αρχείο ***sxesiako_Hotel.jpg*** (επισυνάπτεται και το αρχείο ***sxesiako_Hotel.dia***).

4) Η βάση δεδομένων υλοποιημένη στη postgresql βρίσκεται στο αρχείο ***hotel.sql*** .

5) Τα δεδομένα της πρώτης προσέγγισης βρίσκονται στο αρχείο ***Tables_300K.rar*** και μπορείτε να τα ανακτήσετε από το link: https://www.dropbox.com/s/svt07hsmy1bprxo/Tables_300K.rar .

6) Τα δεδομένα της δεύτερης προσέγγισης βρίσκονται στο φάκελο ***Tables_1M.rar*** και μπορείτε να τα ανακτήσετε από το link: https://www.dropbox.com/s/7y3uc9y4ogmuazj/Table_1M.rar .

7) Τα back up αρχεία ,χωρίς τη χρήση ευρετηρίων, για να γίνουν επαναφορά όπως περιγράφω στον οδηγό εγκατάστασης είναι τα hotel_noindex & hotel1M_noindex που βρίσκονται στο αρχείο back_up_DB_no_index.rar και μπορείτε να τα ανακτήσετε από το link:
https://www.dropbox.com/s/ox6051xh5tusxtd/back_up_DB_no_index.rar

8) Τα back up αρχεία ,με τη χρήση ευρετηρίων, για να γίνουν επαναφορά όπως περιγράφω στον οδηγό εγκατάστασης είναι τα hotel_DB & hotel_DB1M που βρίσκονται στο αρχείο back_up_DB_with_index.rar και μπορείτε να τα ανακτήσετε από το link:
https://www.dropbox.com/s/w2bobomh8hsjo7k/back_up_DB_with_index.rar

14 . Αναφορές

http://www.postgresql.org/docs/9.3/static/using-explain.html#USING-EXPLAIN-BASICS .

http://www.mohawksoft.org/?q=node/56 .

http://revenant.ca/www/postgis/workshop/analysis.html

Βιβλίο: Oreilly PostgreSQL Up and Running July 2012
(https://www.dropbox.com/s/taxp0uni35tzwfp/book.pdf).