Βάσεις Δεδομένων

Αναφορά 4ης εργαστηριακής άσκησης

Ομάδα εργασίας

Λυτρίδης Νίκος:2009030088

Φουντουλάκης Μανόλης:2009030015

Σχεδιασμός και ρύθμιση της Φυσικής Βάσης Δεδομένων

Για να μετρήσουμε καλύτερα τους χρόνους της προσομοίωσης αλλά και για να απλοποιήσουμε λιγάκι τη προσομοίωση μας προχωρήσαμε σε ορισμένες αλλαγές σε σχέση με τη βάση δεδομένων που είχαμε υλοποιήσει στη 3η εργαστηριακή άσκηση.

Αρχικά κρατήσαμε ένα αντίγραφο της προηγούμενης άσκησης(σχήμα,δεδομένα) και στη συνέχεια αφαιρέσαμε το foreign keys και τα triggers και αρχικοποιήσαμε ξανά τη βάση σύμφωνα με τις ζητούμενες προδιαγραφές.

Για τη μέτρηση του χρόνου που διαρκεί ένα query κάναμε χρήση της εντολής της Java System.*currentTimeMillis*() για τα 4 updates μας (U1,U2,U3,U4) καθώς για τα υπόλοιπα 2 Queries μας (Q1,Q2) .

Υλοποίηση του προγράμματος σε java για τη προσομείοση του φόρτου εργασίας.Δημιουργήσαμε μια κλάση `newdtb4` μέσα στην οποία γινόταν όλες οι λειτουργίες.

Αρχικοποιήσεις:

Για όλες τις παρακάτω μεθόδους που αναλύουμε,δημιουργήσαμε μια μεταβλητή π.χ. για την συνάρτηση add_person_client :*CallableStatement cs4=db.conn.prepareCall("{call add_person_client1(?,?,?,?,?,?,?,?,?)}");*

Πιο αναλυτικά οι συναρτήσεις:

U1 :Person-Client-Creditcard

Για την αρχικοποίηση των tables (person-client) φτιάξαμε μια μέθοδο,την *insert_person_many_day()* ,μέσα την οποία εισάγουμε τα δεδομένα στα tables σύμφωνα με τις προδιαγραφές της προηγούμενης άσκησης με τη διαφορά ότι το προσομοιώσαμε για χ μέρες όπως το ζητούσε για το ***U1***.Να διευκρινήσουμε ότι οι εισαγωγές γίνονται στους πίνακες ***client-person-creditcard***.

U2 :Person

Αρχικοποιήσαμε το table φτίαχνοντας την μέθοδο *ins_person()* και εισάγοντας στοιχεία προσώπου σύμφωνα με τις προδιαγραφές της προηγουμένης άσκησης στη περίπωση εισαγωγής employees.Επιπλέον τροποποιήσαμε κατάλληλα έτσι ώστε να εισάγονται σε κάθε κλήση το 5% του συνόλου του ***client,*** για να το πετύχουμε αυτό χρησιμοποιήσαμε τη ν συνάρτηση ***st.executeQuery("select count(documentclient) from client");*** . Με αυτόν τον τρόπο καταφέρνουμε να πάρουμε το πλήθος των ήδη εγγεγραμένων πελατών(μέγεθος το οποίο δεν είναι στατικό) έτσι ώστε να εισάγουμε το αριθμό προσώπων που απαιτήται(5% του συνόλου).’Επειτα προσομοιάσαμε για χ μέρες όπως ζητούσε το ***U2***.

U3:Hotelbooking-Roombooking

Εισάγαμε στοιχεία στα παραπάνω tables φτιάχνοντας τη μέθοδο *hotelbooking()* σύμφωνα με τις προδιαγραφές τις άσκησεις.Προσομοιώσαμε για χ μέρες όπως ζητούσε το ***U3***.Καθημερινά εισάγονται το 20% των εγγεγραμένων πελατών.Για να διαβάσουμε αυτή τη ποσότητα και ανάλογα να βάλουμε το πλήθος των κρατήσεων χρησιμοποιήσαμε όπως και στο U2 την συνάρτηση ***st.executeQuery("select count(documentclient) from client");*** **.** Επίσης για να πετύχουμε την τυχαία επιλογή βάση της οποίας ο πελάτης θα κάνει κράτηση,δηλαδή ξενοδοχείο και δωμάτιο καλέσαμε τη συνάρτηση st1.executeQuery("select idroom from roomss where hotelID='"+hotel+"' "); Αποθηκέυοντας τα idroom σε έναν πίνακα.Στη συνέχεια με τυχαίο τρόπο(χρήση της ZipfGenerator) επιλέξαμε ένα idroom για τον εκάστωτε πελάτη.Στη συνέχεια για το πλήθος διανυκτερέυσεων και το πλήθος κρατήσεων ανά πελάτη λάβαμε υπόψην τις πιθανότητες που μας δόθηκαν και χρησιμοποιώντας την ZipfGenerator το πετύχαμε. Και σε αυτή τη περίπτωση το προσομοιώσαμε για χ μέρες.

U4:Ενημέρωση Hotelbooking-Roombooking

Ενημέρωση(update) tables hotelbooking,roombooking.Αυτό το υλοποίσαμε τη μέθοδο upadate() και δημιουργόντας ένα query βάζωντας το σε ένα string για κάθε ενημέρωση που μας ζητήθηκε να κάνουμε ,μια μεταβλητή τύπου statement και χρησιμοποιώντας την συνάρτηση executeUpdate(query) κάναμε τις ενημερώσεις.Για παραδείγμα όταν θέλουμε να ενημερώσουμε το cancelation date στο hotelbooking κάνουμε το εξής:

```
String query = "UPDATE hotalbooking SET  cancellationdate='"+dat+"'
WHERE idhotelbooking='"+hotelbook+"' ";
Statement stat1 = conn.createStatement();
stat1.executeUpdate(query);
stat1.close();
```

Και σε αυτή τη περίπτωση το προσομοιώσαμε για χ μέρες.

Q1: Εύρεση προσώπων που κατοικούν σε μια συγκεκριμένη πόλη και για τα οποία έχουν γίνει κρατήσεις δωματίων για συγκεκριμένο χρονικό διάστημα διαμονής και από συγκεκριμένο πελάτη.

Καθημερινά εκτελείται τόσες όσες το 20% των εγγεγραμένων πελατών.Και σε αυτή τη περίπτωση κάναμε ότι και στο U2-U3 για την έυρεση του ποσοστού αυτού.
Σύμφωνα με τις απαιτήσεις του ερωτήματος αρχικά δοκιμάσαμε με χρήση του sql editor της postgres να κατασκευάσουμε το κατάλληλο query για να ανακτήσουμε τα κατάλληλα

δεδομένα βάζοντας κατάλλης μεταβλητές.Και στη συνέχεια το προσαρμόσαμε σε java έτσι ώστε το query αυτό να καλείται μέσα από το πρόγραμμα μας.

Αυτό το υλοποιήσαμε ως εξής:
```
Statement stat3= conn.createStatement();//'"+tmp+"'
                ResultSet res1 = stat3.executeQuery("select distinct
idperson from person1,roombooking1 where city='"+location+"' " +
" and person1.idperson=roombooking1.bookedforpersonid and
roombooking1.checkout-roombooking1.checkin='"+meres+"' " +
                        "order by idperson asc; ");


stat3.close();
```

Όπου οι μέρες καιτο location επιλέγονται τυχαία σύμφωνα με το ερώτημα.

Q2: Εντοπισμός προσώπων που έχουν ταυτόχρονη διαμονή σε περισσότερα του ενός δωματίων.

Η λογική υλοποίησης του ερωτήματος είναι ίδια με του q1.Ο κώδικας που χρησιμοποιήσαμε για αυτην την ανάκτηση είναι ο εξής:

```
       Statement stat3= conn.createStatement();//'"+tmp+"'
     ResultSet res1 = stat3.executeQuery("with krathseis as(select
COUNT(idhotelbooking)as " +
            "c1,idperson from person1,hotalbooking1 where
person1.idperson=hotalbooking1.bookedbyclientID " +
            "and hotalbooking1.reservationdate="'+date+'" group by
person1.idperson order by idperson asc)" +
            "select distinct fname,lname ,reservationdate,c1 from
person1,hotalbooking,krathseis where c1>1 and
person1.idperson=krathseis.idperson");
```

Λόγω ότι το ερώτημα είναι πιο περίπλοκο από το προηγουμένο κάναμε χρήση with. Μέσα από τον προσωρινό πίνακα krathseis βλέπουμε το πλήθος κρατήσεων πελάτη για συγκεκριμένη ημερομηνία.Έπειτα με selection προβάλουμε τους πελάτες οι οποίοι έχουνε κάνει ταυτόχρονη διαμονή σε περισσότερα του ενός δωματίου.

Κύρια εργάλεια βελτιστοποίσης που χρησιμοποιήσαμε

Για να βελτιστοποιήσουμε τις εισαγωγές αλλάξαμε το buffer pool και παίρναμε αποτελέσματα.Όσον αφορά τις ανακτήσεις καταλήξαμε στη μέθοδο Partition.Τα αποτελέσματα και πιο συγκεκριμένα οι παράμετροι αναλύονται παρακάτω.

Προσομοίωση του φόρτου εργασίας και βελτιστοποίηση του φυσικου σχεδιασμού

Γενικά κάνουμε τις ενημερώσεις στη βάση μας για τις X,Y,Z μέρες σειριακά αντίστοιχα που μας ζητείται και αμέσως μετά εκτελούμε για μία μέρα τις ζητούμενες ανακτήσεις, όσες φορές απαιτείται για την καθεμιά.

Ως γενικό στόχο αποδοτικότητας για καθεμιά από τις παραπάνω λειτουργίες θέτουμε ένα συγκεκριμμένο όριο Tmax κάθε μεμονωμένης ανάκτησης ή ενημέρωσης και χρόνος μεγαλύτερος του συγκεκριμμένου ορίου ΔΕΝ θα γίνεται αποδεκτός. Σε κάθε στάδιο της ρύθμισης επιλέγουμε να ρυθμίσουμε πρώτα τη λειτουργία με το βραδύτερο συνολικό χρόνο εκτέλεσης ανά ημέρα μέχρις ότου όλες οι μεμονωμένες λειτουργίες εκτελούνται σε χρόνο μικρότερο του Tmax.Ο συνολικός χρόνος εκτέλεσης ανά ημέρα ισούται με το μέσο χρόνο εκτέλεσης της μεμονωμένης λειτουργίας επί το πλήθος των εκτελούμενων λειτουργιών αυτού του τύπου ανά ημέρα.

X=50 ημέρες

Αρχικά προσομοιώσαμε το φόρτο εργασίας για buffer pool(default)=32 mb.

Οι μέσες τιμές για τις ανακτήσεις και τις εισαγωγές είναι οι εξής:

U1:117.64 ms

U2:159.64 ms

U3:1184 ms

U4:2*10^-3 ms

Q1:256.28 ms

Για buffer pool =2mb

U1:117 ms

U2:163.08 ms

U3:1168.9 ms

U4: 0.62 ms

Q1:253.26 ms

Στην συνέχεια χρησιμοποιήσαμε τη μέθοδο Partitioning.Πιο συγκεκριμένα για το Q1

Κάναμε partition στον πίνακα person με βάση το city έτσι ώστε να προκύψουν 10 πίνακες(όσες δηλαδή και οι πόλεις) child τύπου person_city,το σκεπτικό είναι το εξής:αντί να ψάχνει σε όλο το πίνακα person όπου τα tuples είναι διάσπαρτα να φτιάξει και να εισάγη με χρήση κατάλληλων triggers 10 πίνακες όπου θα περιέχουν τα στοιχεία των person σύμφωνα με τη πόλη.Έτσι η ανάκτηση γίνεται σε πίνακες μικρότερου μεγέθους άρα και λιγότερης πολυπλοκότητας.

Παρατηρήσαμε ότι ο μέσος χρόνος για τις εισαγωγές και ενημερώσεις παρέμεινε ο ίδιος .Η διαφορά είναι:

Q1-> 256.28 ms έπεσε στα 186.6 ms.

Για Υ=50 μέρες λειτουργίας ακόμη

Αρχικά προσομοιώσαμε το φόρτο εργασίας για buffer pool(default)=32 mb.

Οι μέσες τιμές για τις ανακτήσεις και τις εισαγωγές είναι οι εξής:

U1:130.6 ms

U2:242,7 ms

U3:3093,12 ms

U4:1.2 ms

Q1:510.58 ms **χωρίς βελτιστοποίηση**

Για buffer pool =2mb

U1:124.72 ms

U2:263.1 ms

U3:3050.96 ms

U4: 1,26 ms

Παρατηρήσαμε ότι με την αλλαγή του buffer pool δεν έχουμε σημαντικές διαφορές για τις εισαγωγές-ενημερώσεις.

Το αποτέλεσμα όμως του q1 μετά τη διαμέριση από 510.58 ms έγινε 385,4 ms.

<u>Για Z=50 μέρες λειτουργίας ακόμη</u>

Αρχικά προσομοιώσαμε το φόρτο εργασίας για <u>buffer pool(default)=32 mb.</u>

Οι μέσες τιμές για τις ανακτήσεις και τις εισαγωγές είναι οι εξής:

U1:127.02 ms

U2:343.82 ms

U3:5756 ms

U4: 1.72 ms

Q1:801,24 ms **χωρίς βελτιστοποίηση**

Στη συνέχεια αλλάξαμε το buffer pool(Default)=16mb

U1:126,7 ms

U2:352,96 ms

U3:5702,42ms

U4: 1.52 ms

Παρατηρήσαμε ότι με την αλλαγή του buffer pool δεν έχουμε σημαντικές διαφορές για τις εισαγωγές-ενημερώσεις.

Το αποτέλεσμα όμως του q1 μετά τη διαμέριση από 801,24 ms έγινε 600,41 ms.

Παρατηρήσεις:

Βλέπουμε ότι όσο αυξάνονται οι μέρες για τις οποίες προσομοιώνουμε το πρόγραμμα μας οι χρόνοι όσο των εισαγωγών τόσο και των ανακτήσεων αυξάνονται.Για τις εισαγωγές U2-U4 είναι λογικό διότι εξαρτάται από τις εισαγωγές που έχουν γίνει στο U1.Οπότε όσες περισσότερες μέρες περνάνει τόσο αυξάνεται το πλήθος εισαγωγών για τα U2-U4,ενώ για το U1 παραμένει στα ίδια επίπεδα διότι καθημερινά εισάγεται συγκεκριμένο πλήθος.Όσον αφορά το Q1 συμβαίνει κάτι αντίστοιχο.Όμως βλέπουμε ότι κάνοντας partition η απόδοση βελτιώνεται για τις εκάστοτε μέρες εκτέλεσης,όμως όχι όσο θα θέλαμε.

Σας παραθέτουμε στο φάκελο το κώδικα σε java που υλοποιήσαμε και τα triggers για το Q1 που υλοποιήσαμε για το partition.