Intervalforespørgsler i SQL

Peter Sestoft (sestoft@dina.kvl.dk)

Symposium for Seriøse Databasestuderende, Frederiksberg, 9. december 1998

- Problemet
- Den ligefremme løsning er ineffektiv
- Indexes considered harmful
- Forsøg på en geometrisk løsning
- En SQL-løsning med asymptotisk lavere kompleksitet

Problemet: find alle overlappende intervaller

Der er givet en relation `projekt` indeholdende en mængde af intervaller.

Skemaet er:

```
(a int, b int, navn int)
```

Givet et interval $[a, b]$, find alle intervaller der overlapper med dette.

Et interval $[a_i, b_i]$ fra relationen overlapper $[a, b]$ hvis

$$a_i \leq b \wedge a \leq b_i$$

Derfor vil følgende SQL-forespørgsel løse problemet:

```
SELECT * FROM projekt WHERE projekt.a <= b AND a <= projekt.b
```

Men denne forespørgsel er langsom

Med N tupler i relationen, vil svartiden i almindelighed være $O(N)$, uanset antallet af tupler n i svaret.

Årsagen er at svaret er en (lille) fællesmængde af to (store) mængder.

Eksperiment: 86452 tilfældige intervaller med startpunkter ligefordelt mellem 0 og 100000 og længder ligefordelt mellem 0 og 1000.

Vælg `a = 50000` og `b = 50001`.

Det tager 0.85 sekunder at udføre denne forespørgsel for 86452 tupler; svaret indeholder 420 tupler.

For 171452 tilfældige intervaller er svartiden 1.60 sekunder; svaret indeholder 856 tupler.

Svartiden er den stabiliserede svartid efter gentagne forespørgsler.

Databaseserveren er PostgreSQL 6.3.2 på en 266MHz PII Dell bærbar.

Forespørgselsplanen benytter table scan.

Indexes considered harmful

Der tilføjes nu B-træ indekser på `a`- og `b`-felterne hver for sig.

Derved stiger svartiden fra 0.85 sekunder til 5.60 sekunder (for 86452 tupler).

PostgreSQL har åbenbart en indbygget query pessimizer.

En geometrisk løsning

PostgreSQL har geometriske datatyper: punkt, rektangel, cirkel, osv, og indekser pã disse.

Et interval $[a_i, b_i]$ kan repræsenteres som punktet (a_i, b_i).

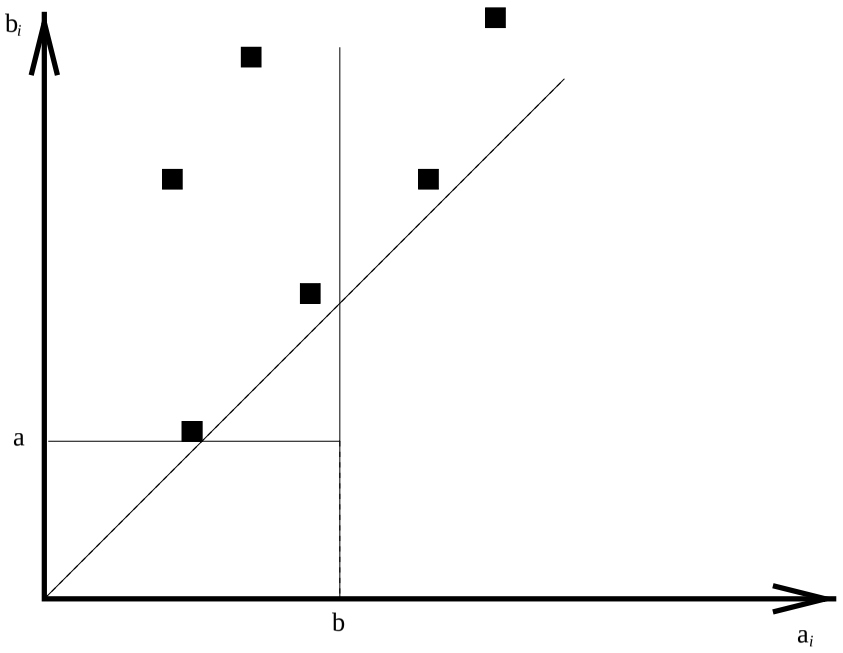
Et interval $[a_i, b_i]$ overlapper med intervallet $[a, b]$ hvis
punktet (a_i, b_i) falder i rektanglet udspændt af $(0, \infty)$ og (b, a)

Vi kan nu repræsentere intervaller som punkter og lave et indeks pã punkterne.

Den geometriske PostgreSQL forespørgsel:

```
SELECT * FROM proj WHERE intv @ box'((0, 100000),(a, b))'
```

udføres faktisk lidt hurtigere end den tilsvarende naive forespørgsel, men det virker ikke som en radikal forbedring.



En anden idé

Hvis

- et interval repræsenteres ved dets midtpunkt og radius (= halve længde):

$$mid_i = \frac{a_i + b_i}{2} \quad \text{og} \quad radius_i = \frac{b_i - a_i}{2}$$

- intervalradierne er begrænsede af $maxradius \geq radius_i$, for $i = 1, \ldots, N$
- der er et B-træ-indeks pã intervalmidtpunkterne

sã kan forespørgslen effektivt begrænses til en delmængde pã denne mãde:

```
SELECT * FROM intv WHERE a+maxradius <= mid AND mid <= b+maxradius
```

Denne forespørgsel besvares faktisk hurtigt, med et index scan.

Svartiden er $O(n)$, lineær i antal tupler n i svaret.

Bemærk at svarer er omtrentligt: en overmængde til det eksakte svar.

Det eksakte svar fãs pã denne mãde (stadig effektivt):

```
SELECT * FROM intv WHERE intv.mid >= a-maxradius AND intv.mid <= b+maxradius
                     AND a <= intv.b AND intv.a <= b
```

Men hvad hvis intervalradierne ikke er begrænsede?

Idé: opdel intervallerne i logaritmisk mange disjunkte radiusklasser:

- Klasse `loggrp` $= 0$ med $radius \leq 1$
- Klasse `loggrp` $= 1$ med $1 < radius \leq 2$
- Klasse `loggrp` $= 2$ med $2 < radius \leq 4$
- Klasse `loggrp` $= 3$ med $4 < radius \leq 8$
- osv.

Nu kan der laves en effektiv approksimativ+eksakt forespørgsel i hver radiusklasse.

Det kræver at der er et fælles B-træ-indeks pã felterne (`loggrp`, `mid`).

Det rigtige svar er nu foreningsmængden af resultaterne for hver klasse.

Et forsøg pã at kombinere radiusklasserne med SQL OR fik query optimizeren til at gã amok: 120 MB RAM og tilsyneladende uendelig køretid.

Svarene fra radiusklasserne skal kombineres med SQL UNION.

Effektiv forespørgsel, for ni radiusklasser 1,...,512

Find overlap med intervallet [50000, 50001]:

```
SELECT * FROM intv WHERE (loggrp = 0 AND mid >= 49999.0 AND mid <=
50002.0 AND start <= 50001 AND slut >= 50000) UNION SELECT * FROM intv
WHERE (loggrp = 1 AND mid >= 49998.0 AND mid <= 50003.0 AND start <=
50001 AND slut >= 50000) UNION SELECT * FROM intv WHERE (loggrp = 2
AND mid >= 49996.0 AND mid <= 50005.0 AND start <= 50001 AND slut >=
50000) UNION SELECT * FROM intv WHERE (loggrp = 3 AND mid >= 49992.0
AND mid <= 50009.0 AND start <= 50001 AND slut >= 50000) UNION SELECT
* FROM intv WHERE (loggrp = 4 AND mid >= 49984.0 AND mid <= 50017.0
AND start <= 50001 AND slut >= 50000) UNION SELECT * FROM intv WHERE
(loggrp = 5 AND mid >= 49968.0 AND mid <= 50033.0 AND start <= 50001
AND slut >= 50000) UNION SELECT * FROM intv WHERE (loggrp = 6 AND mid
>= 49936.0 AND mid <= 50065.0 AND start <= 50001 AND slut >= 50000)
UNION SELECT * FROM intv WHERE (loggrp = 7 AND mid >= 49872.0 AND mid
<= 50129.0 AND start <= 50001 AND slut >= 50000) UNION SELECT * FROM
intv WHERE (loggrp = 8 AND mid >= 49744.0 AND mid <= 50257.0 AND start
<= 50001 AND slut >= 50000) UNION SELECT * FROM intv WHERE (loggrp = 9
AND mid >= 49488.0 AND mid <= 50513.0 AND start <= 50001 AND slut >=
50000)
```

Forespørgslen genereres og udføres af dette program

```
fun query2 (start, slut) =
    let val loggrps = List.tabulate(10, fn i => i)
        val mid0 = (real start + real slut) / 2.0
        val radius0 = (real slut - real start) / 2.0
        fun mkloggrpquery loggrp =
            let val maxdist = pwr2 (real loggrp) + radius0
            in
                String.concat["SELECT * FROM intv WHERE ",
                              "(loggrp = ", Int.toString loggrp,
                              " AND mid >= ", Real.toString(mid0-maxdist),
                              " AND mid <= ", Real.toString(mid0+maxdist),
                              " AND start <= ", Int.toString slut,
                              " AND slut >= ", Int.toString start,
                              ")"]
            end
        fun foldsep sep f []       = []
          | foldsep sep f (x1::xr) =
            f x1 :: List.foldr (fn (x, res) => sep :: f x :: res) [] xr
        val query2 = String.concat (foldsep " UNION " mkloggrpquery loggrps)
        val _ = (print "query2:\n"; print query2; print "\n")
        val res2 = execute pc query2
    in
        print "Antal tupler i resultat: ";
        print (Int.toString (ntuples res2)); print "\n"
    end
```

Eksperimentelle resultater

Opgave: i en relation med 171452 tilfældige intervaller, find overlap med [50000, 50001].

Køretiden er 1.71 sekunder for den naive forespørgsel.

Køretiden er 0.74 sekunder for den radiusklasseinddelte forespørgsel.

Når antallet af tupler i svaret er lille, er den indviklede forespørgsel hurtigst.

Når antallet af tupler i svaret er stort, er den naive forespørgsel hurtigst.