Übungsblatt 3. Aufgabe 3.

Sonntag, 21. November 2021 13:44

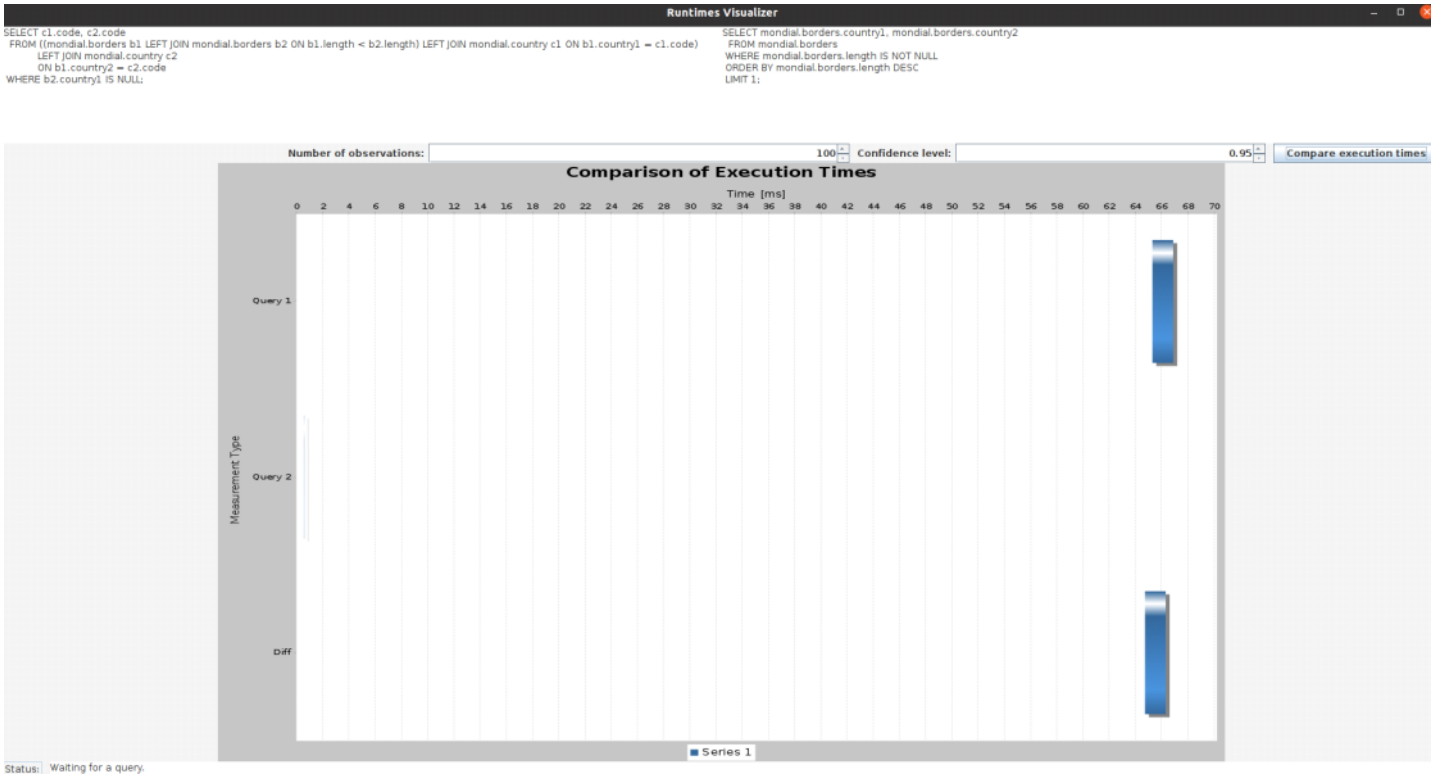
Zwei folgende Anfragen werden für den Test genommen.

Query 1: meine Anfrage aus dem Übungsblatt 2, die 9. Anfrage

```
SELECT c1.code, c2.code
 FROM ((mondial.borders b1 LEFT JOIN mondial.borders b2 ON b1.length < b2.length) LEFT JOIN mondial.country c1 ON
b1.country1 = c1.code)
       LEFT JOIN mondial.country c2
       ON b1.country2 = c2.code
 WHERE b2.country1 IS NULL;
```

Query 2: die verbesserte Version meiner Anfrage

```
SELECT mondial.borders.country1, mondial.borders.country2
 FROM mondial.borders
 WHERE mondial.borders.length IS NOT NULL
 ORDER BY mondial.borders.length DESC
 LIMIT 1;
```



Figur 1. Das Ergebnis des Programms angewendet auf die Query 1 (meine 9. Anfrage aus dem Übungsblatt 2 und die Query 2 (die verbesserte Version)

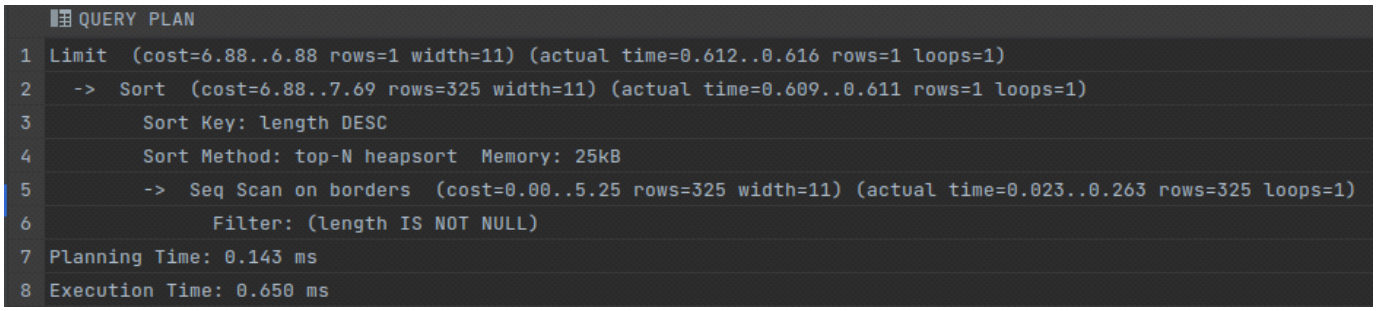
Die Ausführungspläne der Query 1 und 2

```
QUERY PLAN
Nested Loop Left Join  (cost=0.00..1612.71 rows=1 width=6) (actual time=70.052..77.179 rows=1 loops=1)
  Join Filter: ((b1.country2)::text = (c2.code)::text)
  Rows Removed by Join Filter: 194
  ->  Nested Loop Left Join  (cost=0.00..1604.20 rows=1 width=6) (actual time=69.946..77.071 rows=1 loops=1)
        Join Filter: ((b1.country1)::text = (c1.code)::text)
        Rows Removed by Join Filter: 193
        ->  Nested Loop Left Join  (cost=0.00..1595.69 rows=1 width=6) (actual time=69.837..76.961 rows=1 loops=1)
              Join Filter: (b1.length < b2.length)
              Rows Removed by Join Filter: 53022
              Filter: (b2.country1 IS NULL)
              Rows Removed by Filter: 52603
              ->  Seq Scan on borders b1  (cost=0.00..5.25 rows=325 width=11) (actual time=0.017..0.123 rows=325 loops=1)
              ->  Materialize  (cost=0.00..6.88 rows=325 width=8) (actual time=0.000..0.101 rows=325 loops=325)
                    ->  Seq Scan on borders b2  (cost=0.00..5.25 rows=325 width=8) (actual time=0.013..0.260 rows=325 loops=1)
        ->  Seq Scan on country c1  (cost=0.00..5.45 rows=245 width=3) (actual time=0.004..0.054 rows=194 loops=1)
  ->  Seq Scan on country c2  (cost=0.00..5.45 rows=245 width=3) (actual time=0.002..0.052 rows=195 loops=1)
Planning Time: 0.801 ms
Execution Time: 77.232 ms
```

Figur 2. Ausführungsplan der Query 1

Übungsblatt3. Aufgabe 3.

Sonntag, 21. November 2021 15:00



```
QUERY PLAN
Limit  (cost=6.88..6.88 rows=1 width=11) (actual time=0.612..0.616 rows=1 loops=1)
  ->  Sort  (cost=6.88..7.69 rows=325 width=11) (actual time=0.609..0.611 rows=1 loops=1)
        Sort Key: length DESC
        Sort Method: top-N heapsort  Memory: 25kB
        ->  Seq Scan on borders  (cost=0.00..5.25 rows=325 width=11) (actual time=0.023..0.263 rows=325 loops=1)
              Filter: (length IS NOT NULL)
Planning Time: 0.143 ms
Execution Time: 0.650 ms
```

Figur 3. Ausführungsplan der Query 2

DIe genaueren Ergebnisse der Intervalle:

Query 1: 65.29543117539735 66.88140882460264

Query 2: 0.5614942195061863 0.5809257804938137

Diff: 64.72416460278582 66.31025539721419

Erläuterung der Ergebnisse (Hypothese):

Die Query 1 ist deutlich langsamer als die Query 2 (Figur 1). Der Grund ist die Nutzung von zahlreichen JOIN-Operatoren in der ersten Anfrage, wobei in der zweiten gar keine JOINs benutzt werden. Darüber hinaus können wir aus dem Ausführungsplan (Figur 2, Zeile 4) entnehmen, dass das erste JOIN, nämlich ((mondial.borders b1 LEFT JOIN mondial.borders b2 ON b1.length < b2.length) I, das langsamste und aufwändigste ist. Das ist klar, da quasi das Kreuzprodukt, nämlich alle Tupel mit allen in der Relation mondial.borders, berechnet wird, was auch der große Aufwand erzeugt.

In der Query 2 hingegen werden keine JOINs verwendet. Außerdem ist die Sortierungsoperation mit top-N heapsort (Figur 3) mehr optimiert als JOINs. Der Scan mit dem Filter NOT NULL erzeugt keinen zusätzlichen Overhead im Vergleich zu den anderen Operationen wie JOINs oder Sort und ist somit vernachlässigbar.

Dieses Beispiel zeigt, dass die Verwendung von zahlreichen JOINs, insbesondere von JOIN mit sich selbst einen großen Aufwand im Vergleich zu den ORDER BY and WHERE Klauseln erzeugen kann