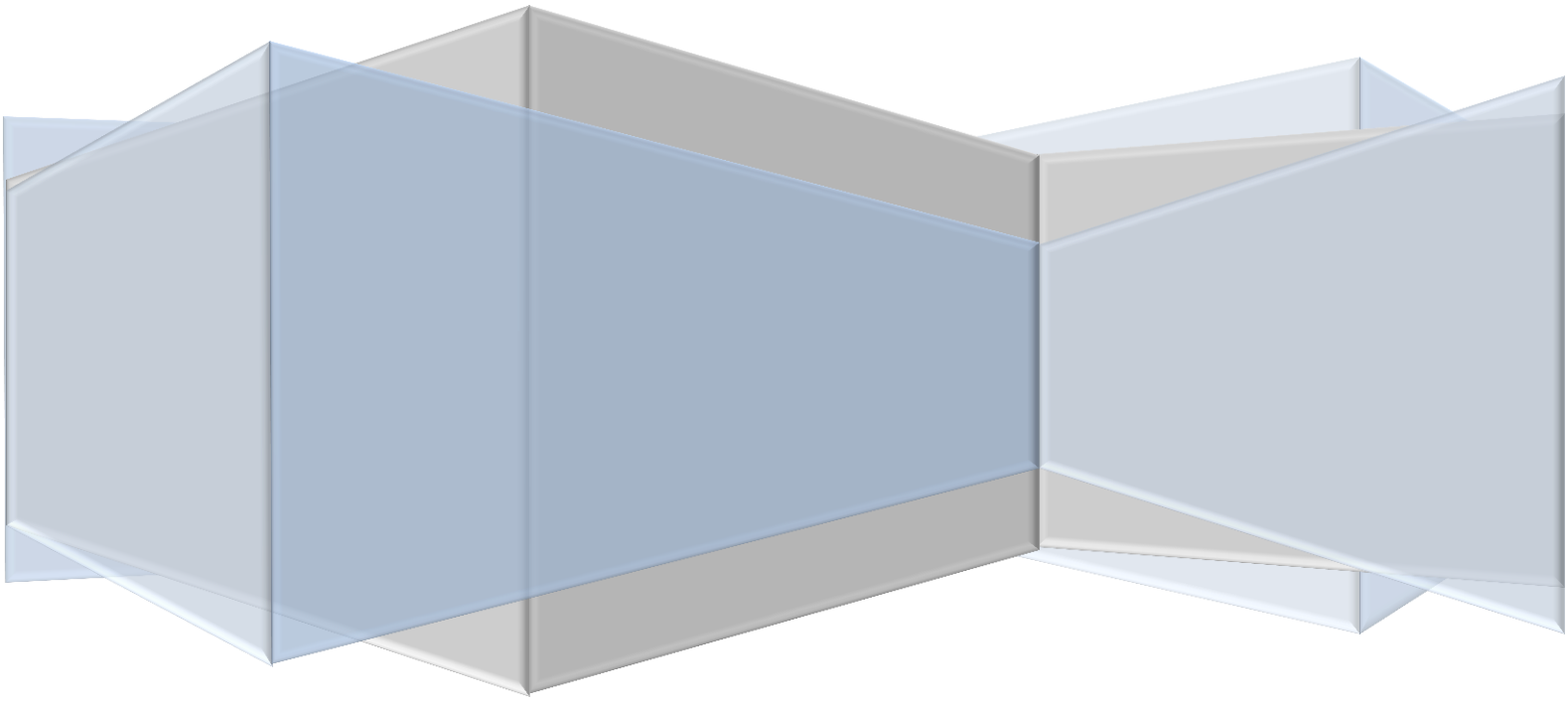
TGM Wien

SQL Tuning

INSY 2014/2015 | Stand: 27.03.2015

Michael Weinberger 4AHIT



Inhalt

Aufgabenstellung

Beschreibung auf Moodle

Dokumentieren Sie alle Tipps aus den vorgestellten Quellen [1,2] mit ausgeführten Queries aus den zur Verfügung gestellten Testdaten [3] in einem PDF-Dokument. Zeigen Sie jeweils die Kosten der optimierten und nicht-optimierten Variante und diskutieren Sie das Ergebnis.

Eine Herausforderung wäre die Schokofabrik-Datenbank mit den generierten 10000 Datensätzen pro Tabelle zu verwenden, dies ist aber nicht Pflicht, ersetzt aber den Einsatz der oben genannten Testdaten.

---

[1] http://beginner-sql-tutorial.com/sql-query-tuning.htm

[2] http://beginner-sql-tutorial.com/sql-tutorial-tips.htm

[3] https://elearning.tgm.ac.at/mod/resource/view.php?id=33104

[4] http://www.borko.at/~mike/Testdaten.zip

Detaillierte Arbeitsaufteilung (Aufwandsabschätzung, Endzeitaufteilung)

Aufwandabschätzung

Diese Aufgabe ist innerhalb von 4-5 Stunden zu schaffen.

Endzeitaufteilung

Derzeit beläuft sicher der Aufwand auf ca. 4 Stunden.

Fazit

Aufgrund meines grippalen Infekts, den ich mir am Samstag auf einer Geburtstagsfeier eingefangen habe, bin ich noch nicht weiter gekommen. Sobald das Fieber besser wird, arbeite ich weiter.

Arbeitsdurchführung

Vor jeder Query wurde ein manuelles *VACUUM* ausgeführt, welches ein ‚Garbage Collector'-Befehl ist, der alte, nicht mehr gebrauchte Kopien einer Reihe löscht, welche das *ANALYZE*-Ergebnis verfälschen. [1]

EXPLAIN zeigt den Ausführplan, sprich wie er erwartet, wie das Ergebnis aussieht mit den jeweiligen Kosten, wobei bei *ANALYZE* die Query auch tatsächlich ausgeführt wird, und die realen Kosten angezeigt werden. [2][3][4]

Verwendet wurde die World Factbook-Datenbank. Alle Tuning-Vorschläge (Überschriften) übernommen aus obenstehenden Quellen.

1) *Eine Query wird schneller abgeschlossen, wenn die eigentlichen Reihen angegeben werden im SELECT statt einem ‚*'.*

Query 1, nicht optimiert:

SELECT * FROM lake;

```
                                QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on lake  (cost=0.00..3.30 rows=130 width=68) (actual time=0.004..0.039 rows=130
loops=1)
 Planning time: 0.122 ms
 Execution time: 0.076 ms
```

Query 2, optimiert:

SELECT name, area, depth, altitude, type, river, longitude, latitude FROM lake;

```
                                QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on lake  (cost=0.00..3.30 rows=130 width=68) (actual time=0.006..0.015 rows=130
loops=1)
 Planning time: 0.037 ms
 Execution time: 0.037 ms
```

Kostenanalyse, Stellungnahme:

Bei beiden Querys wurden 130 Datenreihen abgearbeitet, bei allen 4 Vergleichswerten.
Die Kosten sind normalerweise nicht in ms angegeben, sollten jedoch möglichst proportional zur Zeit liegen. [5]

Die Kosten liegen bei der nicht optimierten Version nur etwas über dem Richtwert, während die optimierte Version diesen weit unterbietet.
Bei der Ausführung ist die 1. Query etwas schneller als der Planungswert, jedoch ist die 2. Query im Gesamten ca. um die Hälfte schneller, und erreicht denselben Zeitwert bei Planung und Ausführung.

2) *Die HAVING-Clause wird verwendet verwendet um die Reihen zu filtern, nachdem alle zurückgeliefert worden sind. Es darf daher nicht für andere Zwecke verwendet werden.*

Query 1, nicht optimiert:

SELECT name, COUNT(name) FROM mountain WHERE name != 'Mulhacen' AND name != 'Grand Ballon' GROUP BY name;

```
                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=7.75..10.11 rows=236 width=11) (actual time=0.147..0.180 rows=236
loops=1)
   Group Key: name
   ->  Seq Scan on mountain  (cost=0.00..6.57 rows=236 width=11) (actual time=0.008..0.063
rows=236 loops=1)
         Filter: (((name)::text <> 'Mulhacen'::text) AND ((name)::text <> 'Grand Ballon'::text))
         Rows Removed by Filter: 2
 Planning time: 0.104 ms
 Execution time: 0.219 ms
```

Query 2, optimiert:

SELECT name, COUNT(name) FROM mountain GROUP BY name HAVING name != 'Mulhacen' AND name != 'Grand Ballon';

```
                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=7.75..10.11 rows=236 width=11) (actual time=0.155..0.192 rows=236
loops=1)
   Group Key: name
   ->  Seq Scan on mountain  (cost=0.00..6.57 rows=236 width=11) (actual time=0.010..0.077
rows=236 loops=1)
         Filter: (((name)::text <> 'Mulhacen'::text) AND ((name)::text <> 'Grand Ballon'::text))
         Rows Removed by Filter: 2
 Planning time: 0.069 ms
 Execution time: 0.228 ms
```

Kostenanalyse, Stellungnahme:

Überall werden 236 Rows angefasst. Die Filter-Bedingung erfasst 2 Datensätze.
Die geplanten Kosten sind bei beiden Querys gleich und recht hoch. In der Ausführung jedoch ist die nicht optimierte Query sogar ein wenig schneller, beide weit unter dem geplanten Wert.

In der Ausführung ist die unoptimierte Version langsamer, verbraucht aber am Ende weniger Zeit als die 2. Query, die trotz einem kleineren Vorbereitungswert ein wenig langsamer ist.

3) *Manchmal hat man mehr als nur eine Subquery in der Haupt-Query. Versuche, diese wo möglich zusammenzufassen.*

Query 1, nicht optimiert:

SELECT name FROM mountain WHERE height = (SELECT MAX(height) FROM mountain) AND latitude = (SELECT MAX(latitude) FROM mountain) AND type = 'volcanic';

```
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=11.97..19.13 rows=1 width=11) (actual time=0.158..0.158 rows=0 loops=1)
   Filter: ((height = $0) AND (latitude = $1) AND ((type)::text = 'volcanic'::text))
   Rows Removed by Filter: 238
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=5.97..5.98 rows=1 width=8) (actual time=0.062..0.062 rows=1 loops=1)
           ->  Seq Scan on mountain mountain_1  (cost=0.00..5.38 rows=238 width=8) (actual time=0.001..0.020 rows=238 loops=1)
   InitPlan 2 (returns $1)
     ->  Aggregate  (cost=5.97..5.98 rows=1 width=8) (actual time=0.055..0.055 rows=1 loops=1)
           ->  Seq Scan on mountain mountain_2  (cost=0.00..5.38 rows=238 width=8) (actual time=0.002..0.018 rows=238 loops=1)
 Planning time: 0.159 ms
 Execution time: 0.215 ms
```

Query 2, optimiert:

SELECT name FROM mountain WHERE (height, latitude) = (SELECT MAX(height), MAX(latitude) FROM mountain) AND type = 'volcanic';

```
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=6.58..13.75 rows=1 width=11) (actual time=0.123..0.123 rows=0 loops=1)
   Filter: ((height = $0) AND (latitude = $1) AND ((type)::text = 'volcanic'::text))
   Rows Removed by Filter: 238
   InitPlan 1 (returns $0,$1)
     ->  Aggregate  (cost=6.57..6.58 rows=1 width=16) (actual time=0.079..0.079 rows=1 loops=1)
           ->  Seq Scan on mountain mountain_1  (cost=0.00..5.38 rows=238 width=16) (actual time=0.001..0.017 rows=238 loops=1)
 Planning time: 0.108 ms
 Execution time: 0.153 ms
```

Kostenanalyse, Stellungnahme:

Wieder sind die Reihen-Werte ident. Ganze 238 Rows werden vom Filter erkannt.
Die Kosten sind angenommen höher als in der Wirklichkeit, um eine große Differenz.
Bei der Zeit in ms gemessen liegt die untere Query natürlich vorne, sowohl beim EXPLAIN-Wert als auch beim ANALYZE-Wert. Die Differenz ist im echten Einsatz schon beachtlich.

4) *Setze EXISTS, IN und JOINS weise ein, junger Padawan.*
IN ist meistens am langsamsten.
IN ist jedoch effizient, wenn die meisten Filterkriterien in der Subquery sind.
EXISTS ist effizient, wenn die meisten Filterkriterien in der Hauptquery sind.

Query 1, nicht optimiert:

SELECT * FROM employees e WHERE religion IN (SELECT name FROM religion);

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Semi Join  (cost=14.21..1246.82 rows=25245 width=105) (actual time=0.624..40.683
rows=25245 loops=1)
   Hash Cond: ((e.religion)::text = (religion.name)::text)
   ->  Seq Scan on employees e  (cost=0.00..810.33 rows=29833 width=105) (actual
time=0.008..7.204 rows=29833 loops=1)
   ->  Hash  (cost=8.54..8.54 rows=454 width=11) (actual time=0.588..0.588 rows=454 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 20kB
         ->  Seq Scan on religion  (cost=0.00..8.54 rows=454 width=11) (actual time=0.012..0.197
rows=454 loops=1)
 Planning time: 1.039 ms
 Execution time: 43.691 ms
```

Query 2, optimiert:

SELECT * FROM employees e WHERE EXISTS (SELECT name FROM religion);

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Result  (cost=0.02..810.35 rows=29833 width=105) (actual time=0.016..9.654 rows=29833 loops=1)
   One-Time Filter: $0
   InitPlan 1 (returns $0)
     ->  Seq Scan on religion  (cost=0.00..8.54 rows=454 width=0) (actual time=0.011..0.011 rows=1
loops=1)
   ->  Seq Scan on employees e  (cost=0.00..810.33 rows=29833 width=105) (actual
time=0.003..2.516 rows=29833 loops=1)
 Planning time: 0.081 ms
 Execution time: 10.825 ms
```

Kostenanalyse, Stellungnahme:

Es wird mit sehr vielen Reihen, 29833 gearbeitet.
Die Kosten unterscheiden sich, die ausgeführten Werte sind ca. um den Faktor 10 schneller.
Die Planning time dauert bei beiden Query nur entsprechend kurz, die Execution time unterscheidet sich um ein Vielfaches. Die optimierte ist um ein Viertel schneller, die nicht optimierte dauert sehr lange 44 ms.

5) *Versuche UNION ALL statt UNION zu verwenden.*

Query 1, nicht optimiert:

SELECT firstname, lastname FROM employees UNION SELECT name, mountains FROM mountain;

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1266.77..1567.48 rows=30071 width=13) (actual time=17.227..24.269
rows=29686 loops=1)
   Group Key: employees.firstname, employees.lastname
   ->  Append  (cost=0.00..1116.42 rows=30071 width=13) (actual time=0.007..7.918 rows=30071
loops=1)
         ->  Seq Scan on employees  (cost=0.00..810.33 rows=29833 width=13) (actual
time=0.007..5.644 rows=29833 loops=1)
         ->  Seq Scan on mountain  (cost=0.00..5.38 rows=238 width=22) (actual time=0.003..0.061
rows=238 loops=1)
 Planning time: 0.063 ms
 Execution time: 25.749 ms
```

Query 2, optimiert:

SELECT firstname, lastname FROM employees UNION ALL SELECT name, mountains FROM mountain;

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..815.71 rows=30071 width=13) (actual time=0.007..7.368 rows=30071 loops=1)
   ->  Seq Scan on employees  (cost=0.00..810.33 rows=29833 width=13) (actual time=0.006..5.255
rows=29833 loops=1)
   ->  Seq Scan on mountain  (cost=0.00..5.38 rows=238 width=22) (actual time=0.002..0.040
rows=238 loops=1)
 Planning time: 0.150 ms
 Execution time: 8.532 ms
```

Kostenanalyse, Stellungnahme:
Bei employees wurden 30.071 Rows angeschaut, bei mountain nur 238.
Die geplanten Kosten explodieren regelrecht, da es sich um eine aufwändige Query handelt. De facto

ist der eigentliche Kostenwert entscheidend (ganz grob Faktor 1000) geringer.
Bei der zweiten Query ist nur einmal grob verschätzt worden im Bereich der Planung, die ausgeführten Werte liegen im verschwindend geringen Rahmen.
Die Planning time sieht bei beiden Querys annehmbar aus, unterscheiden sich jedoch stark bei der Execution time: rund 26 ms bei der nicht optimierten und nur gute 8 ms bei der besseren Variante.

6) *Sei vorsichtig bei Conditions in der WHERE clause, junger Jedi.*

Query 1, nicht optimiert:

SELECT name, mountains, height FROM mountain WHERE height != 1000;

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..5.97 rows=237 width=30) (actual time=0.009..0.068 rows=238
loops=1)
   Filter: (height <> 1000::double precision)
 Planning time: 0.064 ms
 Execution time: 0.085 ms
```

Query 2, optimiert:

SELECT name, mountains, height FROM mountain WHERE height > 1000;

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..5.97 rows=235 width=30) (actual time=0.008..0.060 rows=235
loops=1)
   Filter: (height > 1000::double precision)
   Rows Removed by Filter: 3
 Planning time: 0.057 ms
 Execution time: 0.085 ms
```

Kostenanalyse, Stellungnahme:
In der ersten Query werden 237 Rows verarbeitet, in der zweiten 235.
Die Kosten in der Planung sind exakt gleich, bei den ANALYZE-Ergebnissen ist die optimierte Version leicht schneller.
In der Planning time ist die zweite Query wieder leicht besser, was sich bei der Execution time jedoch nicht zeigt, hier liegen beide in der Zeit gleichauf.

7)

Query 1, nicht optimiert:

SELECT firstname, lastname, email FROM employees WHERE SUBSTR(firstname, 1, 3) = 'Sea';

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..959.50 rows=149 width=41) (actual time=0.259..8.895 rows=24
loops=1)
   Filter: (substr((firstname)::text, 1, 3) = 'Sea'::text)
   Rows Removed by Filter: 29809
 Planning time: 0.051 ms
 Execution time: 8.914 ms
```

Query 2, optimiert:

SELECT firstname, lastname, email FROM employees WHERE firstname LIKE 'Sea%';

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..884.91 rows=3 width=41) (actual time=0.138..4.856 rows=24
loops=1)
   Filter: ((firstname)::text ~~ 'Sea%'::text)
   Rows Removed by Filter: 29809
 Planning time: 0.082 ms
 Execution time: 4.872 ms
```

Kostenanalyse, Stellungnahme:
Bei der optimierten werden viel weniger Datenreihen verwendet, dem Anschein nach wir bei dem 1. Ansatz jede Row überprüft. Der Filter nimmt 29809 Tupel wahr.
Die geplanten Kosten fallen bei beiden recht hoch aus, nach einem ANALYZE wird klar, dass die Kosten weitaus niedriger ausfallen, hier kann die zweite Query seine Stärken ausspielen.
Wie auch bei den Kosten fällt das Endergebnis aus, die Execution time wurde halbiert, die Planning Time ist annähernd gleich.

8)

Query 1, nicht optimiert:

SELECT firstname, lastname, email FROM employees WHERE country || religion = 'BHTBuddhist';

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..959.50 rows=149 width=41) (actual time=0.017..9.103 rows=95
loops=1)
   Filter: (((country)::text || (religion)::text) = 'BHTBuddhist'::text)
   Rows Removed by Filter: 29738
 Planning time: 0.051 ms
 Execution time: 9.122 ms
```

Query 2, optimiert:

SELECT firstname, lastname, email FROM employees WHERE country = 'BHT' AND religion = 'Buddhist';

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..959.50 rows=5 width=41) (actual time=0.013..6.263 rows=95
loops=1)
   Filter: (((country)::text = 'BHT'::text) AND ((religion)::text = 'Buddhist'::text))
   Rows Removed by Filter: 29738
 Planning time: 0.068 ms
 Execution time: 6.284 ms
```

Kostenanalyse, Stellungnahme:
Die Anzahl der verwendeten Datenreihen bei dieser Abfrage konnte drastisch reduziert werden, was natürlich eine Qualitätserhöhung mit sich bringt. Der Filter erkennt 29738 Tupel.
Die erwarteten Kosten sind bei beiden ident gleich hoch, in der Ausführung kommt wieder der Faktor 100 zu tragen. Hier erzielt die optimierte Query den besseren Wert.
In der Planning time ist die erste Query etwas schneller, in der Execution time jedoch kommt der eigentliche Nutzen zum Vorschein: Die optimierte Query ist um ein Drittel schneller.

9)

Query 1, nicht optimiert:

SELECT name, mountains, height FROM mountain WHERE height + 1000 < 2500;

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..6.57 rows=79 width=30) (actual time=0.011..0.061 rows=20
loops=1)
   Filter: ((height + 1000::double precision) < 2500::double precision)
   Rows Removed by Filter: 218
 Planning time: 0.055 ms
 Execution time: 0.075 ms
```

Query 2, optimiert:

SELECT name, mountains, height FROM mountain WHERE height < 1500;

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..5.97 rows=18 width=30) (actual time=0.012..0.053 rows=20
loops=1)
   Filter: (height < 1500::double precision)
   Rows Removed by Filter: 218
 Planning time: 0.060 ms
 Execution time: 0.065 ms
```

Kostenanalyse, Stellungnahme:
In der optimierten Version müssen zumindest vorher gesehen weniger Reihen überprüft werden. 218 Rows werden vom Filter erkannt.
Im Kostenfaktor liegt die zweite Query klar vorne, die Kosten fallen bei beiden Werten geringer aus.
In der Planning time ist die zweite Query höher angesiedelt, in der Ausführung jedoch ist die optimierte Query um 0.01 ms schneller.

10)

Query 1, nicht optimiert:

SELECT name, mountains, height FROM mountain GROUP BY name, mountains, height HAVING height >= MAX(height) AND height <= MIN(height);
Aggregatfunktionen in WHERE sind nicht zulässig!

```
                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=8.36..11.92 rows=238 width=30) (actual time=0.161..0.225 rows=238
loops=1)
   Group Key: name, mountains, height
   Filter: ((height >= max(height)) AND (height <= min(height)))
   ->  Seq Scan on mountain  (cost=0.00..5.38 rows=238 width=30) (actual time=0.006..0.019
rows=238 loops=1)
 Planning time: 0.068 ms
 Execution time: 0.269 ms
```

Query 2, optimiert:

SELECT name, mountains, height FROM mountain GROUP BY name, mountains, height HAVING height BETWEEN MAX(height) AND MIN(height);

```
                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=8.36..11.92 rows=238 width=30) (actual time=0.161..0.220 rows=238
loops=1)
   Group Key: name, mountains, height
   Filter: ((height >= max(height)) AND (height <= min(height)))
   ->  Seq Scan on mountain  (cost=0.00..5.38 rows=238 width=30) (actual time=0.006..0.029
rows=238 loops=1)
 Planning time: 0.064 ms
 Execution time: 0.267 ms
```

Kostenanalyse, Stellungnahme:
Die Anzahl der verwendeten Reihen ist wieder gleich.
Die Kostenplanung fällt ebenso gleichwertig aus, in der Ausführung jedoch kann sich die zweite Query knapp durchsetzen.
Auch im Endergebnis liegen die Werte nah beieinander, wobei die zweite Query in der Planung leicht vorne und auch ausgeführt um 0.002 ms schneller ist.

11)

Query 1, nicht optimiert:

SELECT name, mountains, height FROM mountain WHERE NOT height = 1486;
Auch hier auf der Webseite wieder ein Fehler, NOT muss vorangestellt sein.

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..5.97 rows=237 width=30) (actual time=0.008..0.062 rows=237
loops=1)
   Filter: (height <> 1486::double precision)
   Rows Removed by Filter: 1
 Planning time: 0.057 ms
 Execution time: 0.087 ms
```

Query 2, optimiert:

SELECT name, mountains, height FROM mountain GROUP BY name, mountains, height HAVING height BETWEEN MAX(height) AND MIN(height);

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on mountain  (cost=0.00..5.97 rows=220 width=30) (actual time=0.009..0.065 rows=219
loops=1)
   Filter: (height > 1486::double precision)
   Rows Removed by Filter: 19
 Planning time: 0.058 ms
 Execution time: 0.084 ms
```

Kostenanalyse, Stellungnahme:
Die Anzahl der Reihen kann in der zweiten Query Schritt für Schritt verringert werden. 1 bzw. 19 Datenreihen werden von der Bedingung erfasst.
Die geplanten Kosten sind gleich, die ausgeführten, tatsächlichen Werte gar weniger in der ersten Query. Jedoch müssen auch die Reihen berücksichtigt werden, die ja geringer ausfallen.
So ist das Endergebnis in der Planning time annähernd gleich, jedoch in der Execution time um 0.003 ms schneller.

Quellenangaben:

[1]: http://www.postgresql.org/docs/8.4/static/sql-vacuum.html

[2]: http://www.postgresql.org/docs/9.4/static/sql-explain.html

[3]: http://www.postgresql.org/docs/9.4/static/sql-analyze.html

[4]: http://www.postgresql.org/docs/9.1/static/using-explain.html

[5]: https://wiki.postgresql.org/images/4/45/Explaining_EXPLAIN.pdf