# SQL-Tuning Vergleich

Jakob Greimel

Patrick Wichert

4AHITM

03.06.2016

# SQL-Tuning Vergleich

## Inhalt

# 1. Angabe

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.

Die Übung ist als Gruppenarbeit (2er) zu absolvieren.

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=40861

# 2. SELECT der gewünschten Spalten

Die SQL-Abfrage wird schneller wenn man anstatt "*", die tatsächlich gewünschten Spalten angibt.

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT * FROM *auftrag*; |
| **rows** | 300 |
| **width** | 244 |
| **geschätzte Kosten** | 0.00 .. 13.00 |
| **Zeit** | 0.017 ... 0.022 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *firmenname*, *nummer* FROM *auftrag*; |
| **rows** | 300 |
| **width** | 122 |
| **geschätzte Kosten** | 0.00 .. 13.00 |
| **Zeit** | 0.005 ... 0.011 ms |

Durch die Auswahl der gewünschten Spalten, hat sich die Tabellenbreite um die Hälfte reduziert, dadurch wird auch die Zeit gesenkt.

## 3. SELECT der gewünschten Zeilen

Nur die Zeilen ausgeben die wirklich gebraucht werden.
Je weniger Zeilen abgerufen werden, desto schneller wird die SQL-Abfrage.

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname*, *nachname* FROM *person*; |
| **rows** | 430 |
| **width** | 156 |
| **geschätzte Kosten** | 0.00 .. 14.30 |
| **Zeit** | 0.010 ... 0.014 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname* FROM *person*<br>WHERE *vorname* = 'Wolfgang'; |
| **rows** | 2 |
| **width** | 156 |
| **geschätzte Kosten** | 0.00 .. 15.38 |
| **Zeit** | 0.008 ... 0.010 ms |

Durch das angeben der WHERE Klausel und der gewünschten Zeile wird die Zeit und die Anzahl der Zeilen verringert. Die WHERE Klausel dient als Filter.
Die geschätzten Kosten werden jedoch höher, weil die anderen Zeilen von der WHERE Klausel entfernt werden müssen.

## 4. HAVING Klausel nicht als Filter

Die HAVING Klausel wird benutzt um die Zeilen zu filtern nachdem die ausgewählt wurden.
**Wichtig**: Die HAVING Klausel für keinen anderen Zweck einsetzen.

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *firmenname*, *nummer* FROM *auftrag*<br>GROUP BY *firmenname*, *nummer*<br>HAVING *firmenname* != 'Sweet and Sons' AND *firmenname* != 'Ederle'; |
| **rows** | 198 |
| **width** | 122 |
| **geschätzte Kosten** | 15.98 .. 17.96 |
| **Zeit** | 0.029 ... 0.033 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *firmenname*, *nummer* FROM *auftrag*<br>WHERE *firmenname* != 'Sweet and Sons'  AND *firmenname* != 'Ederle'; |
| **rows** | 297 |
| **width** | 122 |
| **geschätzte Kosten** | 0.00 .. 14.50 |
| **Zeit** | 0.012 ... 0.020 ms |

Durch HAVING wird erst nach dem Auswählen der Zeilen gefiltert, dadurch entstehen höhere Kosten bzw. mehr Zeit.

## 5. Zusammenfassen von Subqueries

Manchmal kann es vorkommen, dass man mehrerer Subqueries in seiner Abfrage hat.
Bei mehreren Subqueries sollte man darauf achten, dass man sie zusammenfasst.

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *firmenname* FROM *auftrag*<br>WHERE *nummer* = (SELECT MAX(*nummer*) FROM *auftrag*)<br>AND *datum* = (SELECT MAX(*datum*) FROM *auftrag*); |
| **rows** | 1 |
| **width** | 118 |
| **geschätzte Kosten** | 27.52 .. 42.02 |
| **Zeit** | 0.030 ... 0.030 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *firmenname* FROM *auftrag*<br>WHERE (*nummer*, *datum*) =<br>(SELECT MAX(*nummer*), MAX(*datum*) FROM *auftrag*); |
| **rows** | 1 |
| **width** | 118 |
| **geschätzte Kosten** | 14.51 .. 29.01 |
| **Zeit** | 0.025 ... 0.025 ms |

Durch einen Subsqueries, erhöht sich die Zeit und die Kosten, da jeder SELECT-Befehl einzeln ausgeführt wird. Durch ein zusammenfügen der beiden Subqueries, muss nur einer ausgeführt werden und das spart Zeit und Kosten

## 6. EXISTS und IN

Benutzen von EXISTS, IN und JOIN um Subqueries zu umgehen und Kosten bzw. Zeit zu sparen.
**IN** hat die langsamere Performance. IN ist jedoch effizient, wenn die meisten Filter Kriterien in der Subquerie enthalten sind.
**EXISTS** hingegen ist effizient, wenn die meisten Filter Kriterien in der Haupt Abfrage enthalten sind.

| **Optimiert IN** | |
|---|---|
| **Abfrage** | *SELECT * FROM auftrag WHERE firmenname IN (SELECT firmenname FROM enthaelt );* |
| **rows** | 150 |
| **width** | 244 |
| **geschätzte Kosten** | 7.49 .. 21.74 |
| **Zeit** | 0.115 ... 0.165 ms |

| **Optimiert EXISTS** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT * FROM *auftrag a* WHERE EXISTS (SELECT * FROM *enthaelt e* WHERE *a.firmenname = e.firmenname);* |
| **rows** | 150 |
| **width** | 224 |
| **geschätzte Kosten** | 7.49 .. 21.49 |
| **Zeit** | 0.118 ... 0.130 ms |

## 7. EXISTS statt DISTINCT

EXISTS sollte man statt DISTINCT bei Joins benutzen die über Tabellen mit one-to-many Beziehung gehen.

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT DISTINCT mit*nummer* FROM bedient b, *mitarbeiter m*<br>WHERE b.mitarbeiter = m.nummer; |
| **rows** | 20 |
| **width** | 4 |
| **geschätzte Kosten** | 64.61 .. 64.81 |
| **Zeit** | 0.074 ... 0.076 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT mit*nummer* FROM bedient b<br>WHERE EXISTS (SELECT nummer FROM mitarbeiter m WHERE m.nummer = b.mitnummer); |
| **rows** | 150 |
| **width** | 4 |
| **geschätzte Kosten** | 60.85 .. 64.34 |
| **Zeit** | 0.34 ... 0.059 ms |

## 8. UNION ALL statt UNION

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer* FROM *person*<br>UNION<br>SELECT *nummer* FROM *mitarbeiter*; |
| **rows** | 2690 |
| **width** | 4 |
| **geschätzte Kosten** | 80.53 .. 107.43 |
| **Zeit** | 0.041 ... 0.049 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer* FROM *person*<br>UNION ALL<br>SELECT *nummer* FROM *mitarbeiter*; |
| **rows** | 2690 |
| **width** | 4 |
| **geschätzte Kosten** | 0.00 .. 46.90 |
| **Zeit** | 0.007 ... 0.017 ms |

UNION ALL als UNION zu verwenden ist schneller, da UNION doppelte Einträge löscht und UNION ALL eine komplette Überprüfung macht, somit ist UNION ALL schneller.

# 9. Conditions in der WHERE Klausel

Man sollte Vorsichtig sein, wenn man Conditions in der WHERE Klausel benutzt.

## 9.1 statt != oder =, < oder >

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer*, *vorname* FROM *person*<br>WHERE *nummer* != 10; |
| **rows** | 429 |
| **width** | 4 |
| **geschätzte Kosten** | 0.00 .. 15.38 |
| **Zeit** | 0.012 ... 0.020 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer*, *vorname* FROM *person*<br>WHERE *nummer* >10; |
| **rows** | 143 |
| **width** | 82 |
| **geschätzte Kosten** | 0.00 .. 15.38 |
| **Zeit** | 0.11 ... 0.019 ms |

## 9.2 LIKE statt =

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname, nachname* FROM *person*<br>WHERE *vorname* = 'Wolfgang'; |
| **rows** | 2 |
| **width** | 156 |
| **geschätzte Kosten** | 0.00 .. 15.38 |
| **Zeit** | 0.014 ... 0.020 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname, nachname* FROM *person*<br>WHERE *vorname* LIKE 'Wolfgang'; |
| **rows** | 2 |
| **width** | 156 |
| **geschätzte Kosten** | 0.00 .. 15.38 |
| **Zeit** | 0.011 ... 0.014 ms |

### 9.3 Keine Rechnungen in der Query

Benutzung von einfachen Operatoren ohne weiteren Rechnungen

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer, vorname* FROM *person* |
| **Optimiert** | |
| **Abfrage** | EXPLAIN ANALYZE SELECT *nummer, vorname* FROM *person* WHERE *nummer* < 20; |
| **rows** | 143 |
| **width** | 82 |
| **geschätzte Kosten** | 0.00 .. 16.45 |
| **Zeit** | 0.008 ... 0.015 ms |

Durch das Rechnen im Filter erhöht sich die Zeit und die Kosten.

## 10. UNIQUE Index für eine Tabelle

Die bestmögliche Art einen UNIQUE Index zu erstellen ist ein Nummerischer Typ der sich automatisch erhöht. Die Suchfunktion wird dadurch schneller als würde man nach einem Wort suchen.

## 11. ORDER BY

ORDER BY mit 2 Spalten

| **Nicht Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname, nachname* FROM *person*; |
| **rows** | 430 |
| **width** | 156 |
| **geschätzte Kosten** | 0.00 .. 14.30 |
| **Zeit** | 0.007 ... 0.014 ms |

| **Optimiert** | |
|---|---|
| **Abfrage** | EXPLAIN ANALYZE SELECT *vorname, nachname* FROM *person* ORDER BY *vorname, nachnname;* |
| **rows** | 430 |
| **width** | 156 |
| **geschätzte Kosten** | 33.11 .. 34.18 |
| **Zeit** | 0.071... 0.073 ms |

Die Abfrage ohne ORDER BY mit 2 Spalten ist schneller da, das Ergebnis nach der Auswahl noch sortiert wird.