Performance einer SQL-Abfrage

Erklärung:

1. **Laufzeit analysieren**: Durch die Ergänzung von `EXPLAIN ANALYSE` erhalten wir Informationen über die Laufzeit der einzelnen Schritte.
2. **Optimieren**: Die gesammelten Informationen zeigen welche Anfragen wir optimieren sollten. Dies geschieht mit einem Hilfsindex (`CREATE INDEX`).

Beispiel: `2. Studenten, die mit Schopenhauer eine Vorlesung gehört haben`

Input:

```
EXPLAIN ANALYSE
SELECT DISTINCT s.Name
FROM Studenten s
JOIN hoeren h1 ON s.MatrNr = h1.MatrNr
JOIN hoeren h2 ON h1.VorlNr = h2.VorlNr
JOIN Studenten schopenhauer ON h2.MatrNr = schopenhauer.MatrNr
WHERE schopenhauer.Name = 'Schopenhauer' AND s.Name != 'Schopenhauer';
```

Output:

```
"QUERY PLAN"
"HashAggregate  (cost=103.38..104.97 rows=159 width=218) (actual time=0.752..0.755
"  Group Key: s.name"
"  Batches: 1  Memory Usage: 40kB"
"  ->  Hash Join  (cost=59.91..102.99 rows=159 width=218) (actual time=0.738..0.744
"        Hash Cond: (h1.matrnr = s.matrnr)"
"        ->  Hash Join  (cost=41.94..84.59 rows=160 width=4) (actual time=0.502..0.5
"              Hash Cond: (h1.vorlnr = h2.vorlnr)"
"              ->  Seq Scan on hoeren h1  (cost=0.00..32.60 rows=2260 width=8) (actu
"              ->  Hash  (cost=41.76..41.76 rows=14 width=4) (actual time=0.261..0.2
"                    Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                    ->  Nested Loop  (cost=4.24..41.76 rows=14 width=4) (actual tim
"                          ->  Seq Scan on studenten schopenhauer  (cost=0.00..14.00
"                                Filter: ((name)::text = 'Schopenhauer'::text)"
"                                Rows Removed by Filter: 7"
```

```
"                           ->  Bitmap Heap Scan on hoeren h2  (cost=4.24..13.77 rows
"                                 Recheck Cond: (matrnr = schopenhauer.matrnr)"
"                                 Heap Blocks: exact=1"
"                                 ->  Bitmap Index Scan on hoeren_pkey  (cost=0.00..4
"                                       Index Cond: (matrnr = schopenhauer.matrnr)"
"        ->  Hash  (cost=14.00..14.00 rows=318 width=222) (actual time=0.224..0.224
"              Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"              ->  Seq Scan on studenten s  (cost=0.00..14.00 rows=318 width=222) (a
"                    Filter: ((name)::text <> 'Schopenhauer'::text)"
"                    Rows Removed by Filter: 1"
"Planning Time: 2.083 ms"
"Execution Time: 0.890 ms"
```

Ergebnis:

- Die Planning Time des gesamten Prozesses ist mit **2.083 ms** sehr hoch!

Optimierung

Wir können durch das Anlegen eines geeigneten Index die **Ausführung beschleunigen**. Ein möglicher Index könnte auf der Spalte ...

1. `MatrNr` der Tabelle `hoeren`,
2. `VorlNr` der Tabelle `hoeren` und
3. `Name` der Tabelle `Studenten` erstellt werden:

Indizes:

```sql
CREATE INDEX idx_hoeren_matrnr ON hoeren (MatrNr);
CREATE INDEX idx_hoeren_vorlnr ON hoeren (VorlNr);
CREATE INDEX idx_studenten_name ON Studenten (Name);
```

Input:

```sql
EXPLAIN ANALYSE
SELECT DISTINCT s.Name
FROM Studenten s
JOIN hoeren h1 ON s.MatrNr = h1.MatrNr
JOIN hoeren h2 ON h1.VorlNr = h2.VorlNr
JOIN Studenten schopenhauer ON h2.MatrNr = schopenhauer.MatrNr
WHERE schopenhauer.Name = 'Schopenhauer' AND s.Name != 'Schopenhauer';
```

Output:

```
"QUERY PLAN"
"Unique  (cost=4.71..4.72 rows=1 width=218) (actual time=0.553..0.557 rows=3 loops=1
"  ->  Sort  (cost=4.71..4.72 rows=1 width=218) (actual time=0.552..0.554 rows=3 lo
"        Sort Key: s.name"
"        Sort Method: quicksort  Memory: 25kB"
"        ->  Hash Join  (cost=3.50..4.70 rows=1 width=218) (actual time=0.532..0.53
"              Hash Cond: (h1.matrnr = s.matrnr)"
"              ->  Hash Join  (cost=2.31..3.51 rows=2 width=4) (actual time=0.318..
"                    Hash Cond: (h1.vorlnr = h2.vorlnr)"
"                    ->  Seq Scan on hoeren h1  (cost=0.00..1.13 rows=13 width=8) (
"                    ->  Hash  (cost=2.29..2.29 rows=2 width=4) (actual time=0.085.
"                          Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                          ->  Hash Join  (cost=1.11..2.29 rows=2 width=4) (actual
"                                Hash Cond: (h2.matrnr = schopenhauer.matrnr)"
"                                ->  Seq Scan on hoeren h2  (cost=0.00..1.13 rows=1
"                                ->  Hash  (cost=1.10..1.10 rows=1 width=4) (actual
"                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB
"                                      ->  Seq Scan on studenten schopenhauer  (cos
"                                            Filter: ((name)::text = 'Schopenhauer'
"                                            Rows Removed by Filter: 7"
"              ->  Hash  (cost=1.10..1.10 rows=7 width=222) (actual time=0.208..0.2
"                    Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                    ->  Seq Scan on studenten s  (cost=0.00..1.10 rows=7 width=222
"                          Filter: ((name)::text <> 'Schopenhauer'::text)"
"                          Rows Removed by Filter: 1"
"Planning Time: 0.394 ms"
"Execution Time: 0.775 ms"
```

Ergebnis:

- Die Planning Time des gesamten Prozesses ist mit **0.394 ms** deutlich geringer als vor der Optimierung!