SQL HINTS, TIPS, TRICKS AND TUNING

Susanne Ebrecht

PGCon 2013, Ottawa
May 2013
• Susanne Ebrecht
• Diploma in Computer Sciences
• Open Source activity 1996
• Expert in Datenbases and Localisation / Globalisation
• Member of SQL Standard Committee
RULES

• Interposed Question are Welcome
• Twitter @miraceesusanne
• No Individual Consulting
• Slides have no informative value outside the talk
WORK_MEM

- SHOW work_mem;
- SET work_mem='64MB';
- Per Session
- (total RAM / max_connections) / query execution steps
Data Definition Language
CREATE, ALTER, DROP

Data Modification Language
INSERT, UPDATE, DELETE

Data Query Language
SELECT

Data Control Language
GRANT, REVOKE

Transaction Control Language
START TRANSACTION, SAVEPOINT, COMMIT, ROLLBACK
SQL TUNING

• DML Tuning thwarts DQL

• DQL Tuning thwarts DMS
NORMALIZATION

• Too much joins apply the brakes
• 3NF is a good start
• Wise Denormalisation
B-TREE
B-TREE
CREATE INDEX ... WITH (FILLFACTOR=n)

ALTER INDEX ... SET (FILLFACTOR=n)

10 < n < 100
INDEX BLOAT

DROP INDEX name
CREATE INDEX name ON ...;
REINDEX name;
CREATE INDEX CONCURRENTLY name_neu ON ...;
DROP INDEX name_alt;
ALTER INDEX name_neu RENAME TO name_alt;
INDEX USAGE

• Index on person(name, given_name)

  ... WHERE (name, given_name)=('Miller', 'Solveig') ...

• Index on person(name) and Index on person(given_name)

  ... WHERE name='Miller' AND given_name='Solveig' ...
JOINS

OUTER JOINS

INNER JOIN

SELECT * FROM A JOIN B ON A.id=B.id;
SELECT * FROM A, B WHERE A.id=B.id;
SELECT A.* FROM A WHERE A.id IN (SELECT B.id FROM B);

LEFT JOIN
SELECT * FROM A LEFT JOIN B ON A.id=B.id
WHERE B.id IS NULL

RIGHT JOIN
SELECT * FROM A RIGHT JOIN B ON B.id=A.id
WHERE A.id IS NULL

FULL JOIN
SELECT * FROM A FULL JOIN B ON A.id=B.id
WHERE A.id IS NULL OR B.id is NULL
CROSS JOINS

Each element with each element ....
Table A: (1,a), (3,b), (5,c)
Table B: (1,x), (2,y), (3,z)

```
knolle=# SELECT * FROM a CROSS JOIN b;

<table>
<thead>
<tr>
<th>i_a</th>
<th>wert_a</th>
<th>i_b</th>
<th>wert_b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>2</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>3</td>
<td>z</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>2</td>
<td>y</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>3</td>
<td>z</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>2</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>3</td>
<td>z</td>
</tr>
</tbody>
</table>

(9 rows)
```
WHAT’S CORRECT?

ABC

AB

A

B

C

ACB

AC

A

B

C

CBA

CB

A

B

C

AB

AC

CB
WHAT’S FASTER?

SELECT * FROM a
  WHERE a.id NOT IN (SELECT id FROM b);

SELECT a.* FROM a LEFT JOIN b ON a.id=b.id
  WHERE b.id IS NULL;
CORRELATED SUBSELECT?

```
SELECT town, violation, amount
FROM ticket_officer AS toff1
WHERE amount = (  
    SELECT max(amount)  
    FROM ticket_officer AS toff2  
    WHERE toff2.town=toff1.town  
);  

=> needs 250 ms
```
WITH max_by_town AS ( 
  SELECT town, max(amount) as total 
  FROM ticket_office 
  GROUP BY town 
) 
SELECT town, violation, amount 
FROM ticket_office 
WHERE town, amount IN ( 
  SELECT town, total 
  FROM max_by_town 
) ;

=> needs 3 ms
WITH income AS ( 
    SELECT town, sum(amount) AS total
    FROM ticket_office
    GROUP BY town
),
top AS ( 
    SELECT town 
    FROM income 
    WHERE total > (SELECT avg(total) FROM income)
)
SELECT town, violation, sum(quantity) AS quantity, sum(amount) AS total 
FROM ticket_office 
WHERE town in (SELECT town FROM top) 
GROUP BY town, ticket;
WITH RECURSIVE meine(n) AS 
( 
  VALUES(1) 
  UNION ALL 
  SELECT n+1 FROM meine WHERE n < 100 
) 
SELECT SUM(n) FROM meine;
PLANER

• EXPLAIN - strategy

• EXPLAIN ANALYZE - strategy and execution
ticket=# EXPLAIN SELECT t.town, tckt.violation, SUM(to.amount) AS total
FROM town AS t JOIN ticket_office AS to ON t.shortcut=to.town JOIN ticket AS tckt ON to.violation=tckt.violation
GROUP BY t.town, tckt.violation ORDER BY total DESC LIMIT 10;

QUERY PLAN

<table>
<thead>
<tr>
<th>Cost</th>
<th>Rows</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>4878.07</td>
<td>10</td>
<td>69</td>
</tr>
<tr>
<td>4878.10</td>
<td>10</td>
<td>69</td>
</tr>
<tr>
<td>4894.18</td>
<td>25245</td>
<td>69</td>
</tr>
<tr>
<td>3827.64</td>
<td>25245</td>
<td>69</td>
</tr>
<tr>
<td>3890.75</td>
<td>25245</td>
<td>69</td>
</tr>
<tr>
<td>561.98</td>
<td>25245</td>
<td>69</td>
</tr>
<tr>
<td>71.17</td>
<td>1020</td>
<td>32</td>
</tr>
<tr>
<td>73.72</td>
<td>1020</td>
<td>32</td>
</tr>
<tr>
<td>20.20</td>
<td>1020</td>
<td>32</td>
</tr>
<tr>
<td>490.81</td>
<td>4950</td>
<td>67</td>
</tr>
<tr>
<td>503.19</td>
<td>4950</td>
<td>67</td>
</tr>
<tr>
<td>33.50</td>
<td>4950</td>
<td>67</td>
</tr>
<tr>
<td>187.05</td>
<td>4950</td>
<td>67</td>
</tr>
<tr>
<td>0.00</td>
<td>1020</td>
<td>32</td>
</tr>
<tr>
<td>19.90</td>
<td>990</td>
<td>48</td>
</tr>
<tr>
<td>21.00</td>
<td>1000</td>
<td>37</td>
</tr>
</tbody>
</table>

EXPLAIN

Susanne Ebrecht

SQL hints, tips, tricks and tuning
EXPLAIN

- cost = estimated operation time; from_ms .... to_ms
- row = estimated number of found rows
- width = estimated row width given in Byte
EXPLAIN ANALYZE

ticket=# EXPLAIN ANALYZE SELECT t.town, tckt.violation, SUM(to.amount) AS total
FROM town AS t JOIN ticket_office AS to ON t.shortcut=to.town JOIN ticket AS tckt ON to.violation=tckt.violation
GROUP BY t.town, tckt.violation ORDER BY total DESC LIMIT 10;

QUERY PLAN

Limit (cost=4878.07..4878.10 rows=10 width=69) (actual time=26.814..26.815 rows=10 loops=1)
  ->  Sort (cost=4878.07..4941.18 rows=25245 width=69) (actual time=26.812..26.812 rows=10 loops=1)
      Sort Key: (sum(to.amount))
      Sort Method: top-N heapsort  Memory: 25kB
  ->  GroupAggregate (cost=3827.64..4332.54 rows=25245 width=69) (actual time=25.631..26.597 rows=256 loops=1)
      ->  Sort (cost=3827.64..3890.75 rows=25245 width=69) (actual time=25.617..25.712 rows=1000 loops=1)
          Sort Key: t.town, tckt.violation
          Sort Method: quicksort  Memory: 125kB
      ->  Merge Join (cost=561.98..945.76 rows=25245 width=69) (actual time=10.094..12.171 rows=1000 loops=1)
          Merge Cond: (tckt.violation = to.violation)
          ->  Sort (cost=71.17..73.72 rows=1020 width=32) (actual time=0.102..0.103 rows=13 loops=1)
              Sort Key: tckt.violation
              Sort Method: quicksort  Memory: 25kB
          ->  Seq Scan on ticket tckt  (cost=0.00..20.20 rows=1020 width=32) (actual time=0.007..0.674 rows=1000 loops=1)
      ->  Hash Join (cost=33.50..187.05 rows=4950 width=67) (actual time=1.684..2.487 rows=1000 loops=1)
          Hash Cond: ((t.shortcut)::text = (to.town)::text)
          ->  Seq Scan on town t  (cost=0.00..19.90 rows=990 width=48) (actual time=0.003..0.014 rows=13 loops=1)
          ->  Hash (cost=21.00..21.00 rows=1000 width=37) (actual time=1.659..1.659 rows=1000 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 69kB
      ->  Hash Join (cost=33.50..187.05 rows=4950 width=67) (actual time=1.684..2.487 rows=1000 loops=1)
          Hash Cond: ((t.shortcut)::text = (to.town)::text)
          ->  Seq Scan on town t  (cost=0.00..19.90 rows=990 width=48) (actual time=0.003..0.014 rows=21 loops=1)
          ->  Hash (cost=21.00..21.00 rows=1000 width=37) (actual time=1.659..1.659 rows=1000 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 69kB
      ->  Seq Scan on ticket_office to (cost=0.00..21.00 rows=1000 width=37) (actual time=0.007..0.674 rows=1000 loops=1)

Total runtime: 26.920 ms
After running ANALYZE

ticket=# EXPLAIN ANALYZE SELECT t.town, tckt.violation, SUM(to.amount) AS total
FROM town AS t JOIN ticket_office AS to ON t.shortcut=to.town JOIN ticket AS tckt ON to.violation=tckt.violation
GROUP BY t.town, tckt.violation ORDER BY total DESC LIMIT 10;

QUERY PLAN

Limit (cost=67.39..67.42 rows=10 width=44) (actual time=5.586..5.590 rows=10 loops=1)
  ->  Sort (cost=67.39..68.08 rows=273 width=44) (actual time=5.584..5.586 rows=10 loops=1)
      Sort Key: (sum(to.amount))
      Sort Method: top-N heapsort  Memory: 25kB
  ->  HashAggregate (cost=58.77..61.49 rows=273 width=44) (actual time=5.080..5.240 rows=256 loops=1)
        ->  Hash Join (cost=2.77..51.27 rows=1000 width=44) (actual time=0.084..2.812 rows=1000 loops=1)
               Hash Cond: (to.violation = tckt.violation)
               ->  Hash Join (cost=1.47..36.22 rows=1000 width=44) (actual time=0.048..1.716 rows=1000 loops=1)
                      Hash Cond: ((to.town)::text = (t.shortcut)::text)
                      ->  Seq Scan on ticket_office to  (cost=0.00..21.00 rows=1000 width=37) (actual time=0.008..0.326 rows=1000 loops=1)
                      ->  Hash (cost=1.21..1.21 rows=21 width=12) (actual time=0.028..0.028 rows=21 loops=1)
                             Buckets: 1024  Batches: 1  Memory Usage: 1kB
                      ->  Seq Scan on town t  (cost=0.00..1.21 rows=21 width=12) (actual time=0.003..0.014 rows=21 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 1kB
               ->  Hash (cost=1.13..1.13 rows=13 width=30) (actual time=0.027..0.027 rows=13 loops=1)
                      Buckets: 1024  Batches: 1  Memory Usage: 1kB
               ->  Seq Scan on ticket tckt  (cost=0.00..1.13 rows=13 width=30) (actual time=0.008..0.015 rows=13 loops=1)

Total runtime: 5.686 ms
ACTUAL

- time = needed operation time; from_ms …. to_ms
- row = Number of found rows
- loops = Number of executions per operation
PGADMIN III
BREAKDOWN

- (cost=0.00..19.90 rows=990 width=48) (actual time=0.003..0.011 rows=21 loops=1)
  - ANALYZE or STATISTIC TARGET

- (actual time=10.081..15.764 rows=1000 loops=651)
  - Think about logic, Redesign Query, CTE (Common Table Expression)

- (actual time=25.617..12425.712 rows=1000 loops=1)
  - Think about logic, Redesign Query, Indexes
STATISTIC

• Random Sample

• postgresql.conf: default_statistic_target = 100

• ALTER TABLE ... ALTER COLUMN ... SET STATISTIC value;
ANALYZE

• SQL Command
• PostgreSQL also allows British: ANALYSE
• Frequency analysis
• Statistic tables like pg_class
• autovacuum includes autoanalyze
<table>
<thead>
<tr>
<th></th>
<th>exclusive</th>
<th>inclusive</th>
<th>rows x</th>
<th>rows</th>
<th>loops</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.003</td>
<td>26.815</td>
<td>↑1.0</td>
<td>10</td>
<td>1</td>
<td>Limit (cost=4878.07..4878.10 rows=10 width=69) (actual time=26.814..26.815 rows=10 loops=1)</td>
</tr>
<tr>
<td>2</td>
<td>0.215</td>
<td>26.812</td>
<td>↑2524.5</td>
<td>10</td>
<td>1</td>
<td>Sort (cost=4878.07..4941.18 rows=25245 width=69) (actual time=26.812..26.812 rows=10 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Key: (sum(kv.betrag))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Method: top-N heapsort Memory: 25kB</td>
</tr>
<tr>
<td>3</td>
<td>0.885</td>
<td>26.597</td>
<td>↑98.6</td>
<td>256</td>
<td>1</td>
<td>GroupAggregate (cost=3827.64..4332.54 rows=25245 width=69) (actual time=25.631..25.597 rows=256 loops=1)</td>
</tr>
<tr>
<td>4</td>
<td>13.541</td>
<td>25.712</td>
<td>↑25.2</td>
<td>1000</td>
<td>1</td>
<td>Sort (cost=3827.64..3890.75 rows=25245 width=69) (actual time=25.617..25.712 rows=1000 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Key: s.stadt, k.verstoff</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Method: quicksort Memory: 125kB</td>
</tr>
<tr>
<td>5</td>
<td>2.007</td>
<td>12.171</td>
<td>↑25.2</td>
<td>1000</td>
<td>1</td>
<td>Merge Join (cost=561.98..945.76 rows=25245 width=69) (actual time=10.094..12.171 rows=1000 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Merge Cond: (k.verstoff = kv.verstoff)</td>
</tr>
<tr>
<td>6</td>
<td>0.089</td>
<td>0.103</td>
<td>↑78.5</td>
<td>13</td>
<td>1</td>
<td>Sort (cost=71.17..73.72 rows=1020 width=32) (actual time=0.102..0.103 rows=13 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Key: k.verstoff</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Method: quicksort Memory: 25kB</td>
</tr>
<tr>
<td>7</td>
<td>0.014</td>
<td>0.014</td>
<td>↑78.5</td>
<td>13</td>
<td>1</td>
<td>Seq Scan on knoelichen k (cost=0.00..20.20 rows=1020 width=32) (actual time=0.009..0.014 rows=13 loops=1)</td>
</tr>
<tr>
<td>8</td>
<td>7.574</td>
<td>10.061</td>
<td>↑5.0</td>
<td>1000</td>
<td>1</td>
<td>Sort (cost=490.81..503.19 rows=4950 width=67) (actual time=9.986..10.061 rows=1000 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Key: kv.verstoff</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sort Method: quicksort Memory: 125kB</td>
</tr>
<tr>
<td>9</td>
<td>0.817</td>
<td>2.487</td>
<td>↑5.0</td>
<td>1000</td>
<td>1</td>
<td>Hash Join (cost=33.50..187.05 rows=4950 width=67) (actual time=1.684..2.487 rows=1000 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hash Cond: (s.kennzahlen::text = (kv.stadt)::text)</td>
</tr>
<tr>
<td>10</td>
<td>0.011</td>
<td>0.011</td>
<td>↑47.1</td>
<td>21</td>
<td>1</td>
<td>Seq Scan on stadt s (cost=0.00..19.90 rows=990 width=48) (actual time=0.003..0.011 rows=21 loops=1)</td>
</tr>
<tr>
<td>11</td>
<td>0.985</td>
<td>1.659</td>
<td>↑1.0</td>
<td>1000</td>
<td>1</td>
<td>Hash (cost=21.00..21.00 rows=1000 width=37) (actual time=1.659..1.659 rows=1000 loops=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Buckets: 1024 Batches: 1 Memory Usage: 69kB</td>
</tr>
<tr>
<td>12</td>
<td>0.674</td>
<td>0.674</td>
<td>↑1.0</td>
<td>1000</td>
<td>1</td>
<td>Seq Scan on knoelchenvergabe kv (cost=0.00..21.00 rows=1000 width=37) (actual time=0.007..0.674 rows=1000 loops=1)</td>
</tr>
</tbody>
</table>
### Per node type stats

<table>
<thead>
<tr>
<th>node type</th>
<th>count</th>
<th>sum of times</th>
<th>% of query</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupAggregate</td>
<td>1</td>
<td>0.885 ms</td>
<td>3.3 %</td>
</tr>
<tr>
<td>Hash</td>
<td>1</td>
<td>0.985 ms</td>
<td>3.7 %</td>
</tr>
<tr>
<td>Hash Join</td>
<td>1</td>
<td>0.817 ms</td>
<td>3.0 %</td>
</tr>
<tr>
<td>Limit</td>
<td>1</td>
<td>0.003 ms</td>
<td>0.0 %</td>
</tr>
<tr>
<td>Merge Join</td>
<td>1</td>
<td>2.007 ms</td>
<td>7.5 %</td>
</tr>
<tr>
<td>Seq Scan</td>
<td>3</td>
<td>0.699 ms</td>
<td>2.6 %</td>
</tr>
<tr>
<td>Sort</td>
<td>4</td>
<td>21.419 ms</td>
<td>79.9 %</td>
</tr>
</tbody>
</table>

### Per table stats

<table>
<thead>
<tr>
<th>Table name</th>
<th>Scan count</th>
<th>Total time</th>
<th>% of query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scan type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>count</td>
<td>sum of times</td>
<td>% of table</td>
</tr>
<tr>
<td>knoellichen</td>
<td>1</td>
<td>0.014 ms</td>
<td>0.1 %</td>
</tr>
<tr>
<td>Seq Scan</td>
<td>1</td>
<td>0.014 ms</td>
<td>100.0 %</td>
</tr>
<tr>
<td>knoellchenvergabe</td>
<td>1</td>
<td>0.674 ms</td>
<td>2.5 %</td>
</tr>
<tr>
<td>Seq Scan</td>
<td>1</td>
<td>0.674 ms</td>
<td>100.0 %</td>
</tr>
<tr>
<td>stadt</td>
<td>1</td>
<td>0.011 ms</td>
<td>0.0 %</td>
</tr>
<tr>
<td>Seq Scan</td>
<td>1</td>
<td>0.011 ms</td>
<td>100.0 %</td>
</tr>
</tbody>
</table>
THANKS FOR LISTENING