SQL HINTS, TIPS, TRICKS AND TUNING

Susanne Ebrecht

PGCon 2013, Ottawa

May 2013

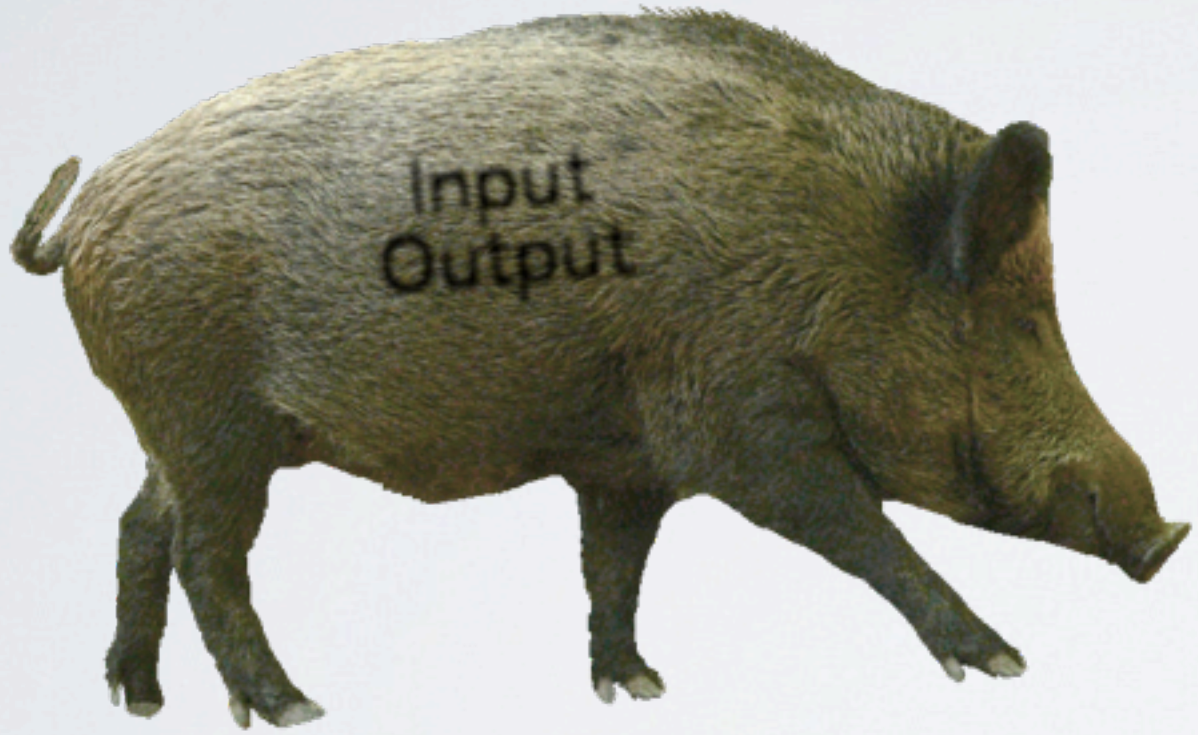
SPEAKER

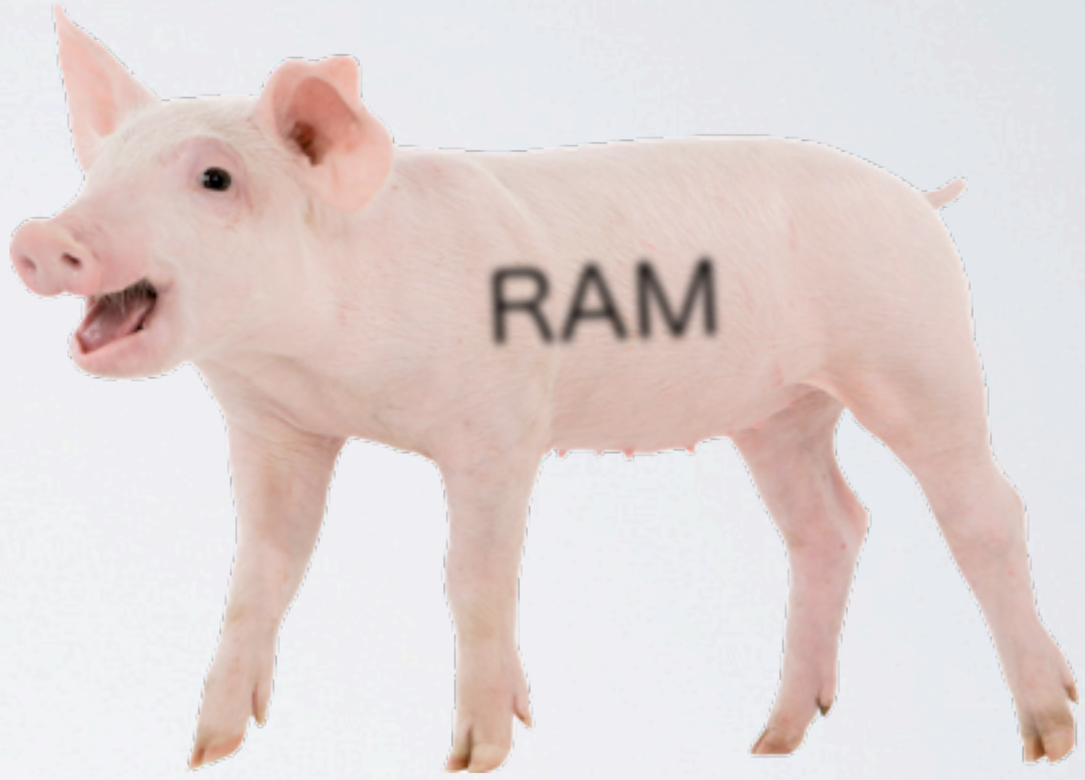
- Susanne Ebrecht
- Diploma in Computer Sciences
- Open Source activity 1996
- Expert in Datenbases and
  Localisation / Globalisation
- Member of SQL Standard Committee

RULES

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

NICKS





WORK_MEM

- SHOW work_mem;
- SET work_mem=‘64MB‘;
- Per Session
- (total RAM / max_connections) / query execution steps

SQL

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

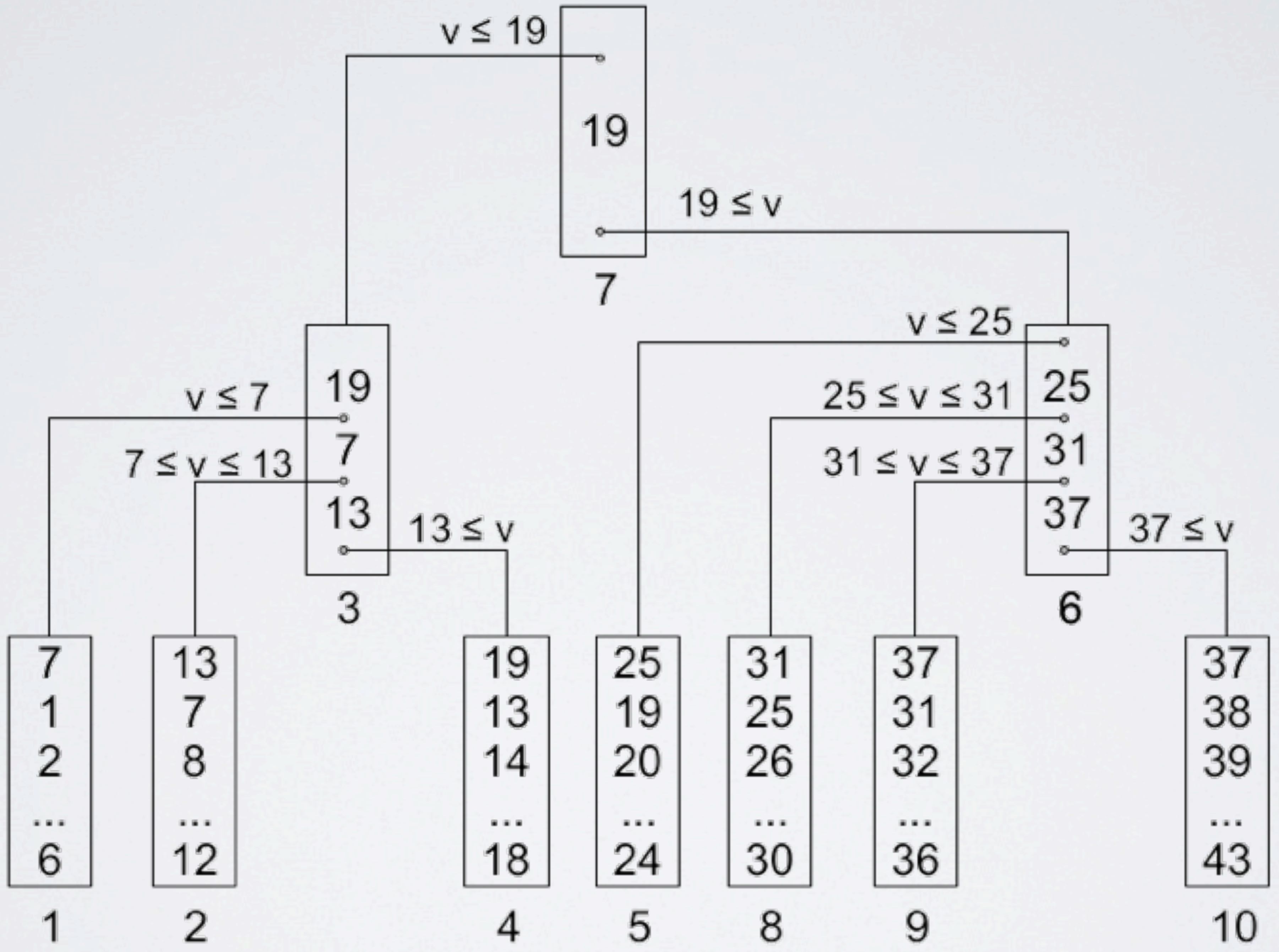
SQL TUNING

- DML Tuning thwarts DQL
- DQL Tuning thwarts DMS

NORMALIZATION

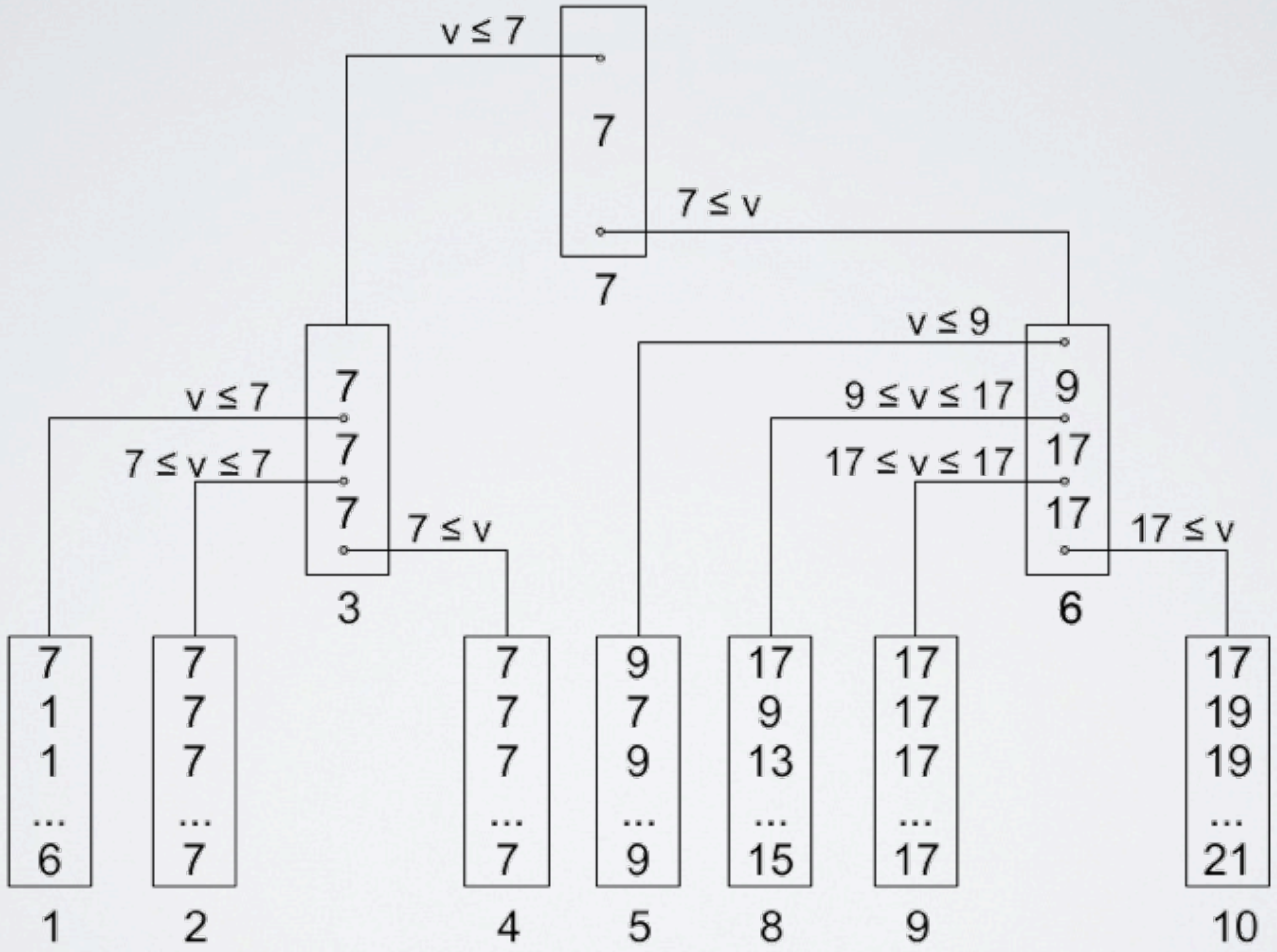
- Too much joins apply the brakes
- 3NF is a good start
- Wise Denormalisation

B-TREE



v ≤ 19
19
19 ≤ v
7
v ≤ 7
19
7
7 ≤ v ≤ 13
13
13 ≤ v
3
v ≤ 25
25 ≤ v ≤ 31
25
31
31 ≤ v ≤ 37
37
37 ≤ v
6
7 1 2 ... 6
1
13 7 8 ... 12
2
19 13 14 ... 18
4
25 19 20 ... 24
5
31 25 26 ... 30
8
37 31 32 ... 36
9
37 38 39 ... 43
10

B-TREE


v ≤ 7
7
7 ≤ v
7
v ≤ 7
7
7 ≤ v ≤ 7
7
7
7 ≤ v
3
v ≤ 9
9 ≤ v ≤ 17
9
17 ≤ v ≤ 17
17
17
17 ≤ v
6
7 1 1 … 6
1
7 7 7 … 7
2
7 7 7 … 7
4
9 7 9 … 9
5
17 9 13 … 15
8
17 17 17 … 17
9
17 19 19 … 21
10

FILLFACTOR

```
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

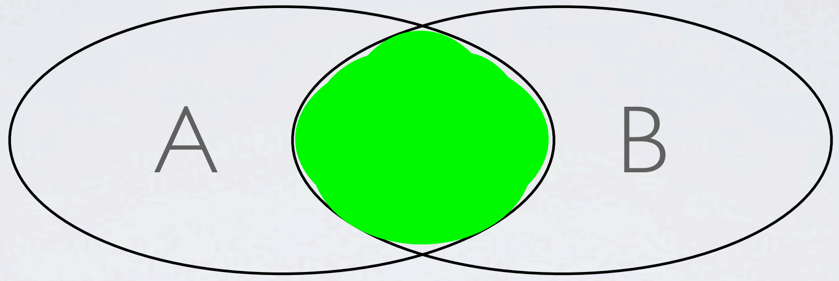
- Index on person(name, given_name)

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

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

  ... WHERE name=‘Miller‘ AND given_name=‘Solveig‘ ...

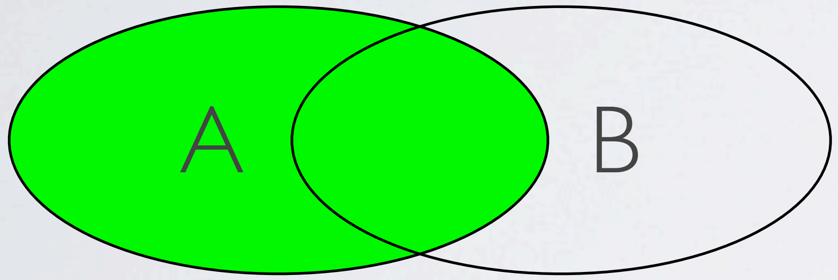
JOINS



INNER JOIN

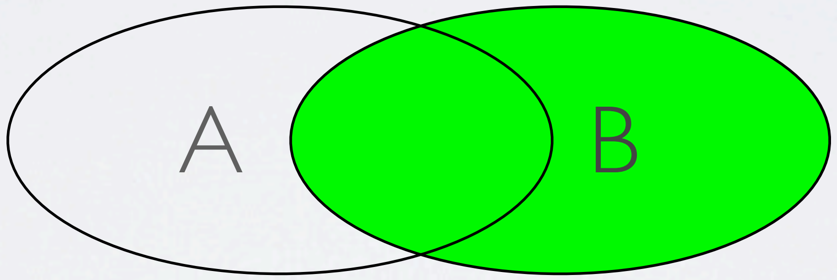
```
SELECT * FROM A JOIN B ON A.id=B.id;
SELECT * FROM A, B WHERE A.id=B.id;
SELECT A.* FROM A WHERE A.id IN
(SELECT B.id FROM B);
```

OUTER JOINS



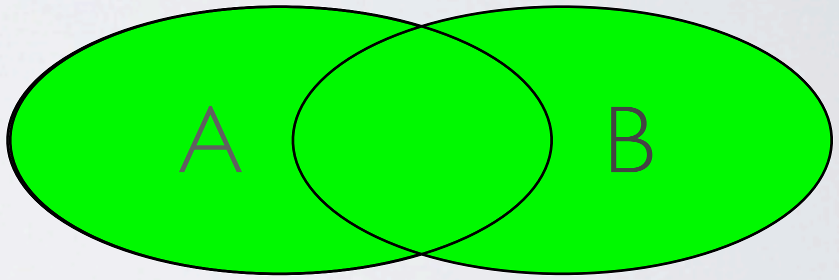
LEFT JOIN

```
SELECT * FROM A LEFT JOIN B
ON A.id=B.id
```



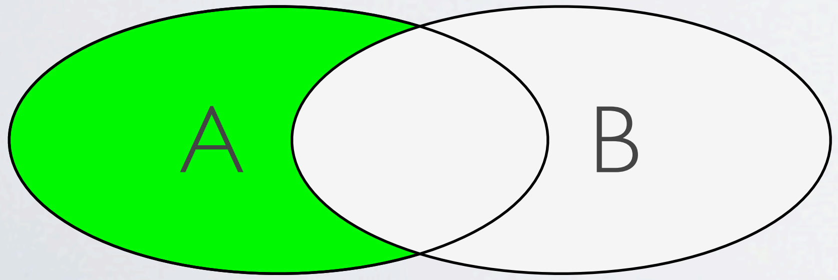
RIGHT JOIN

```
SELECT * FROM A RIGHT JOIN B
ON B.id=A.id
```

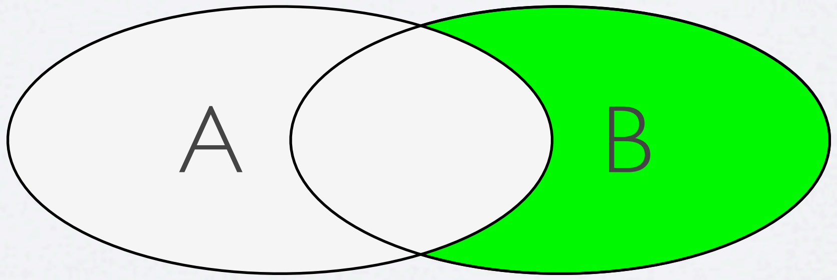


FULL JOIN

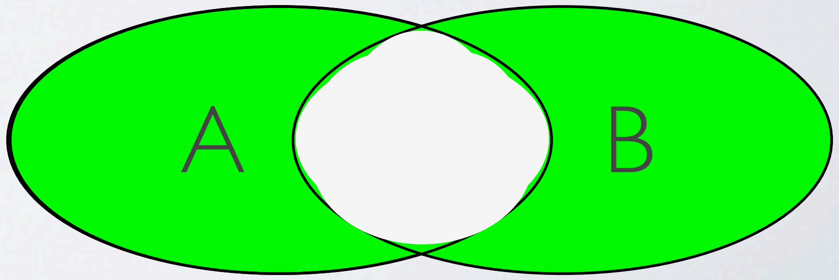
```
SELECT * FROM A FULL JOIN B
ON A.id=B.id
```



```
WHERE B.id IS NULL
```



```
WHERE A.id IS NULL
```



```
WHERE A.id IS NULL OR B.id is NULL
```

CROSS JOINS

Each element with each element ....

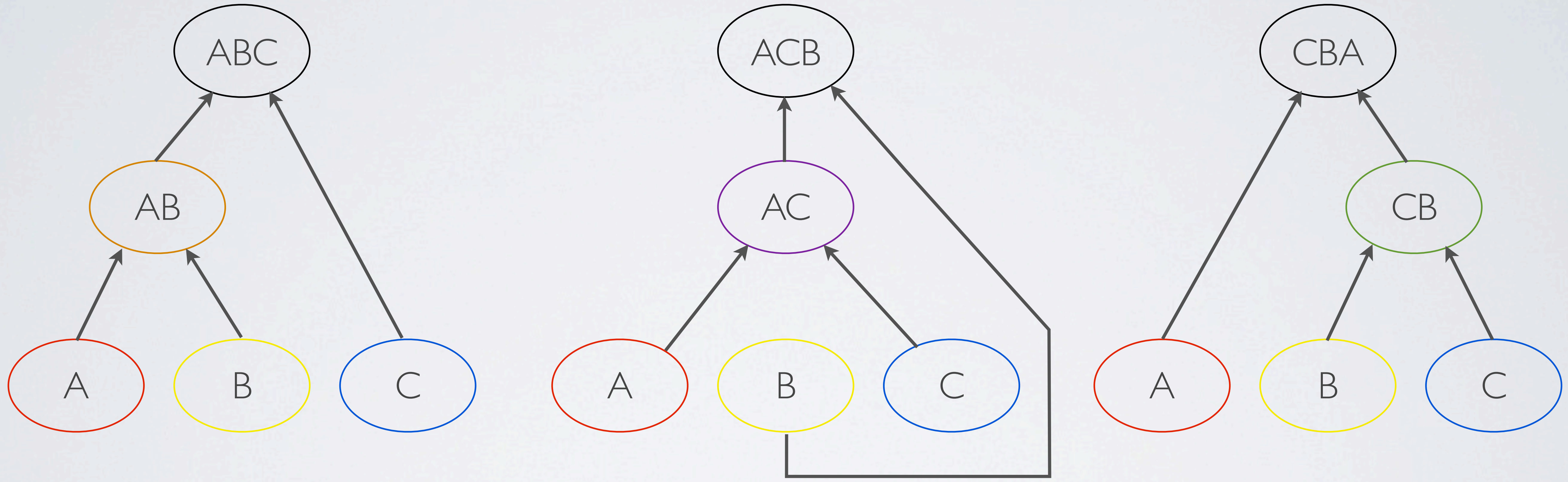
Table A: (1,a), (3,b), (5,c)

Table B: (1,x), (2,y), (3,z)

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

 i_a | wert_a | i_b | wert_b
-----+--------+-----+--------
   1 | a      |   1 | x
   1 | a      |   2 | y
   1 | a      |   3 | z
   3 | b      |   1 | x
   3 | b      |   2 | y
   3 | b      |   3 | z
   5 | c      |   1 | x
   5 | c      |   2 | y
   5 | c      |   3 | z
(9 rows)
```

WHAT'S CORRECT?



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

CTE

Common Table Expression

```
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

CTE

Common Table Expression

```
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;
```

CTE

Common Table Expression

```
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

EXPLAIN

```
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
------------------------------------------------------------------------------------------------------------------
 Limit  (cost=4878.07..4878.10 rows=10 width=69)
   ->  Sort  (cost=4878.07..4941.18 rows=25245 width=69)
         Sort Key: (sum(to.amount))
         ->  GroupAggregate  (cost=3827.64..4332.54 rows=25245 width=69)
               ->  Sort  (cost=3827.64..3890.75 rows=25245 width=69)
                     Sort Key: t.town, tckt.violation
                     ->  Merge Join  (cost=561.98..945.76 rows=25245 width=69)
                           Merge Cond: (tckt.violation = to.violation)
                           ->  Sort  (cost=71.17..73.72 rows=1020 width=32)
                                 Sort Key: tckt.violation
                                 ->  Seq Scan on ticket tckt  (cost=0.00..20.20 rows=1020 width=32)
                           ->  Sort  (cost=490.81..503.19 rows=4950 width=67)
                                 Sort Key: to.violation
                                 ->  Hash Join  (cost=33.50..187.05 rows=4950 width=67)
                                       Hash Cond: ((t.shortcut)::text = (to.town)::text)
                                       ->  Seq Scan on town t  (cost=0.00..19.90 rows=990 width=48)
                                       ->  Hash  (cost=21.00..21.00 rows=1000 width=37)
                                             ->  Seq Scan on ticket_office to  (cost=0.00..21.00 rows=1000 width=37)
```

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=0.009..0.014 rows=13 loops=1)
                           ->  Sort  (cost=490.81..503.19 rows=4950 width=67) (actual time=9.986..10.061 rows=1000 loops=1)
                                 Sort Key: to.violation
                                 Sort Method: quicksort  Memory: 125kB
                                 ->  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..0.011 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

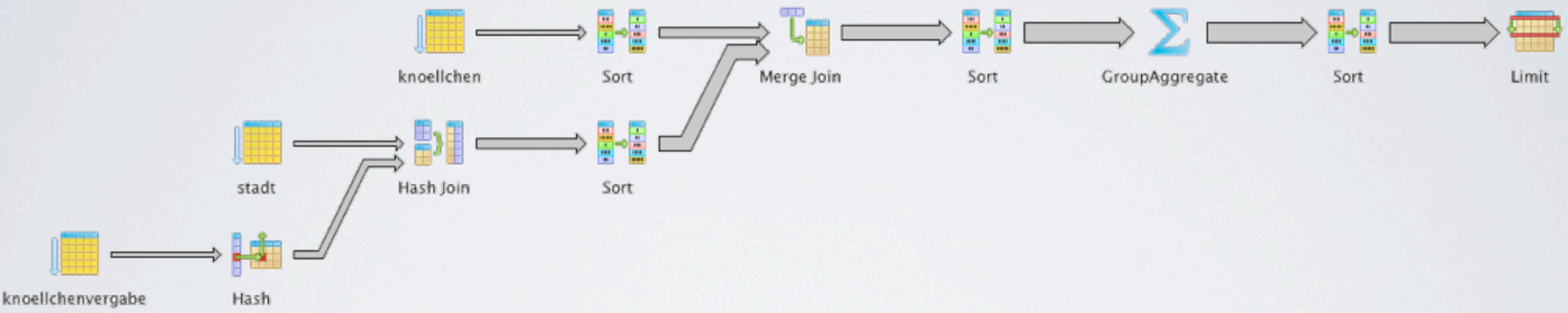
```
ticket=# EXPLAIN ANALYZE SELECT t.town, tckt.violation, SUM(to.amount) AS total
FROM town AS t JOIN ticket_office AS to ON t.shortcut=to.town JOIN ticket AS tckt ON to.violation=tckt.violation
GROUP BY t.town, tckt.violation ORDER BY total DESC LIMIT 10;

                                                          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)
                    ->  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


knoellchen
Sort
Merge Join
Sort
GroupAggregate
Sort
Limit
stadt
Hash Join
Sort
knoellchenvergabe
Hash

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

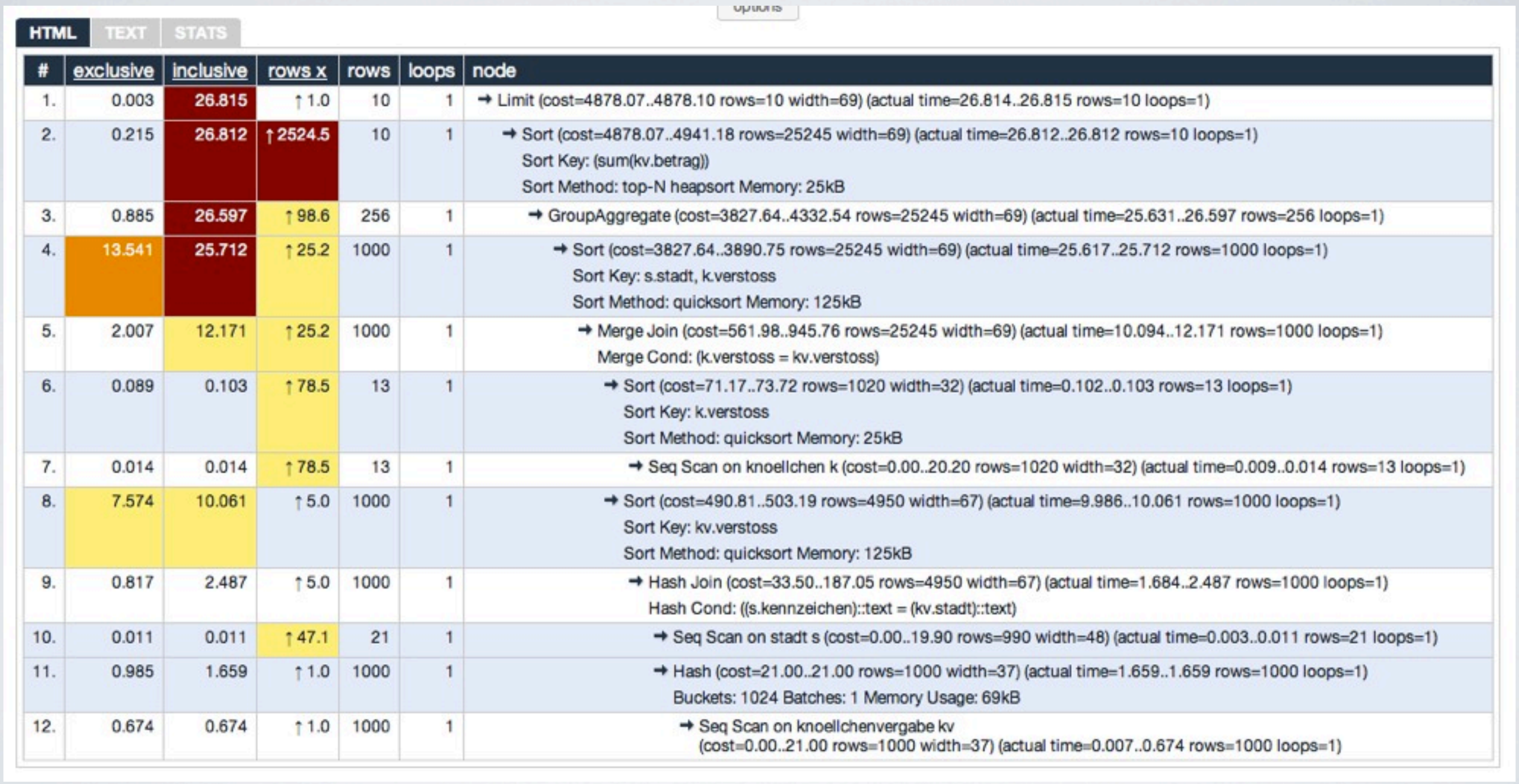
STATISTIC

- 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

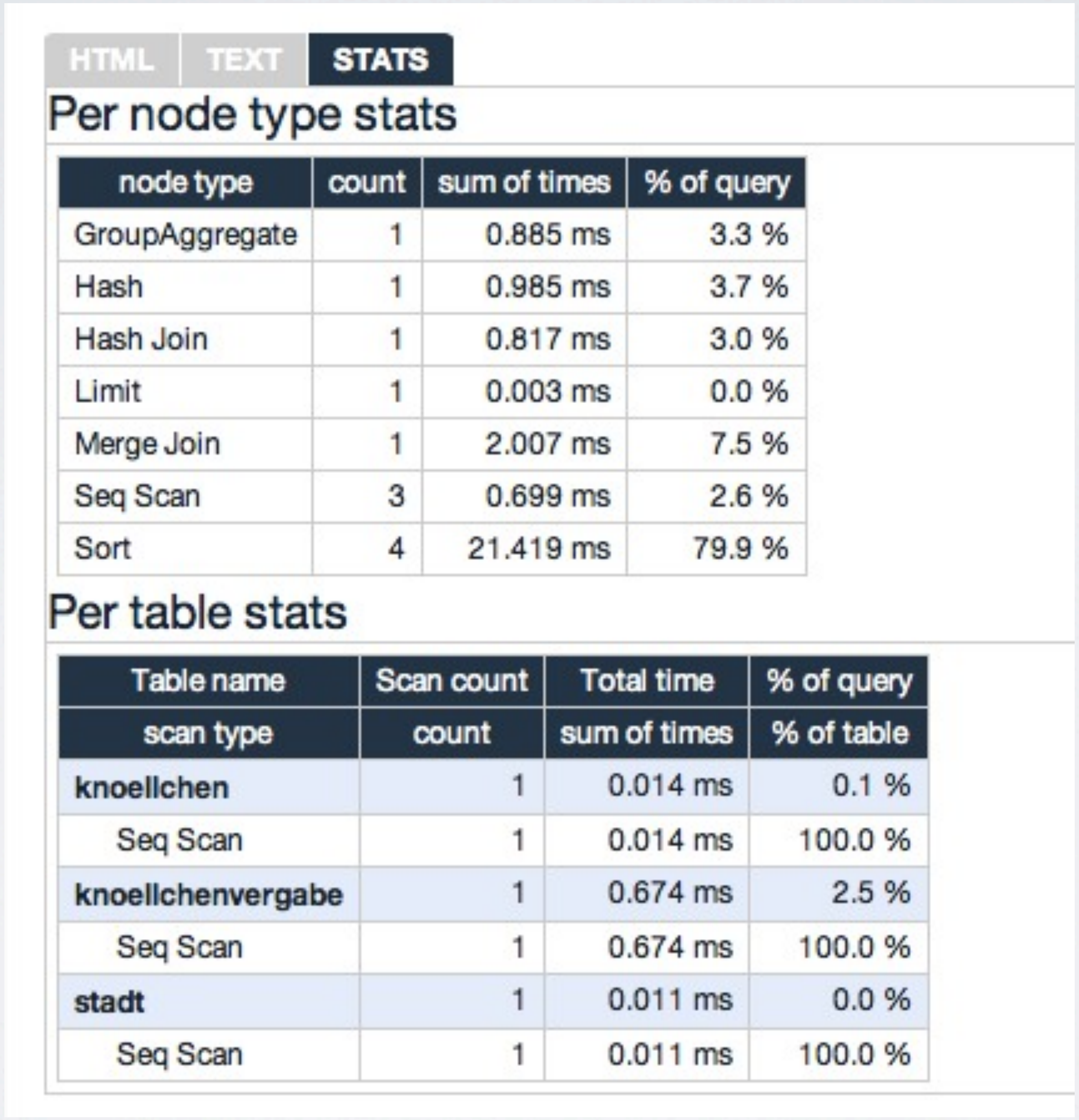
EXPLAIN.DEPESZ.COM



#	exclusive	inclusive	rows x	rows	loops	node
1.	0.003	26.815	↑ 1.0	10	1	→ Limit (cost=4878.07..4878.10 rows=10 width=69) (actual time=26.814..26.815 rows=10 loops=1)
2.	0.215	26.812	↑ 2524.5	10	1	→ Sort (cost=4878.07..4941.18 rows=25245 width=69) (actual time=26.812..26.812 rows=10 loops=1) Sort Key: (sum(kv.betrag)) Sort Method: top-N heapsort Memory: 25kB
3.	0.885	26.597	↑ 98.6	256	1	→ GroupAggregate (cost=3827.64..4332.54 rows=25245 width=69) (actual time=25.631..26.597 rows=256 loops=1)
4.	13.541	25.712	↑ 25.2	1000	1	→ Sort (cost=3827.64..3890.75 rows=25245 width=69) (actual time=25.617..25.712 rows=1000 loops=1) Sort Key: s.stadt, k.verstoss Sort Method: quicksort Memory: 125kB
5.	2.007	12.171	↑ 25.2	1000	1	→ Merge Join (cost=561.98..945.76 rows=25245 width=69) (actual time=10.094..12.171 rows=1000 loops=1) Merge Cond: (k.verstoss = kv.verstoss)
6.	0.089	0.103	↑ 78.5	13	1	→ Sort (cost=71.17..73.72 rows=1020 width=32) (actual time=0.102..0.103 rows=13 loops=1) Sort Key: k.verstoss Sort Method: quicksort Memory: 25kB
7.	0.014	0.014	↑ 78.5	13	1	→ Seq Scan on knoellchen k (cost=0.00..20.20 rows=1020 width=32) (actual time=0.009..0.014 rows=13 loops=1)
8.	7.574	10.061	↑ 5.0	1000	1	→ Sort (cost=490.81..503.19 rows=4950 width=67) (actual time=9.986..10.061 rows=1000 loops=1) Sort Key: kv.verstoss Sort Method: quicksort Memory: 125kB
9.	0.817	2.487	↑ 5.0	1000	1	→ Hash Join (cost=33.50..187.05 rows=4950 width=67) (actual time=1.684..2.487 rows=1000 loops=1) Hash Cond: ((s.kennzeichen)::text = (kv.stadt)::text)
10.	0.011	0.011	↑ 47.1	21	1	→ Seq Scan on stadt s (cost=0.00..19.90 rows=990 width=48) (actual time=0.003..0.011 rows=21 loops=1)
11.	0.985	1.659	↑ 1.0	1000	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
12.	0.674	0.674	↑ 1.0	1000	1	→ Seq Scan on knoellchenvergabe kv (cost=0.00..21.00 rows=1000 width=37) (actual time=0.007..0.674 rows=1000 loops=1)

Hubert Lubaczewski, Nickname: depesz

EXPLAIN.DEPESZ.COM



HTML TEXT STATS

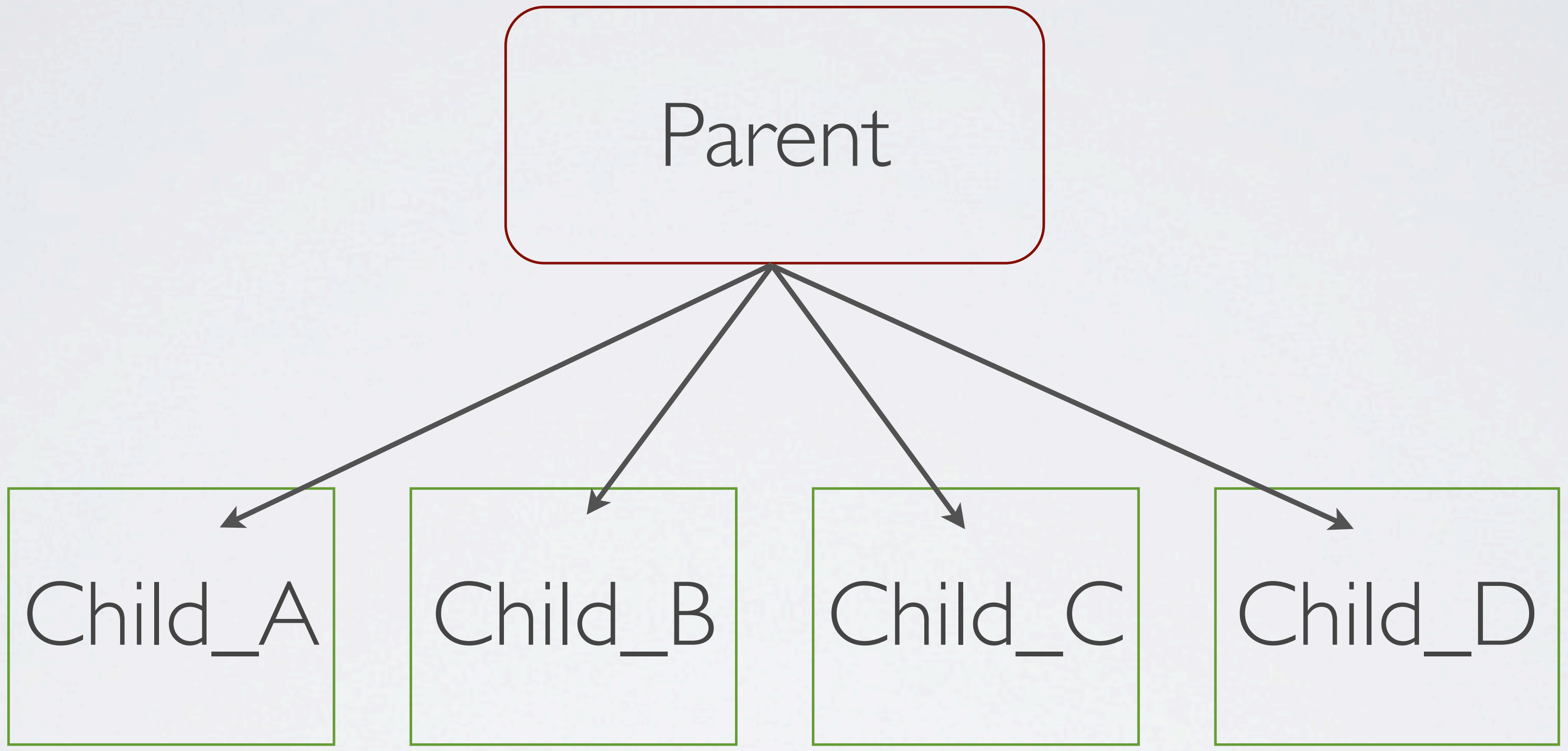
Per node type stats

node type	count	sum of times	% of query
GroupAggregate	1	0.885 ms	3.3 %
Hash	1	0.985 ms	3.7 %
Hash Join	1	0.817 ms	3.0 %
Limit	1	0.003 ms	0.0 %
Merge Join	1	2.007 ms	7.5 %
Seq Scan	3	0.699 ms	2.6 %
Sort	4	21.419 ms	79.9 %

Per table stats

Table name	Scan count	Total time	% of query
scan type	count	sum of times	% of table
knoellchen	1	0.014 ms	0.1 %
Seq Scan	1	0.014 ms	100.0 %
knoellchenvergabe	1	0.674 ms	2.5 %
Seq Scan	1	0.674 ms	100.0 %
stadt	1	0.011 ms	0.0 %
Seq Scan	1	0.011 ms	100.0 %

PARTITIONING



THANKS FOR LISTENING