EXPLAIN

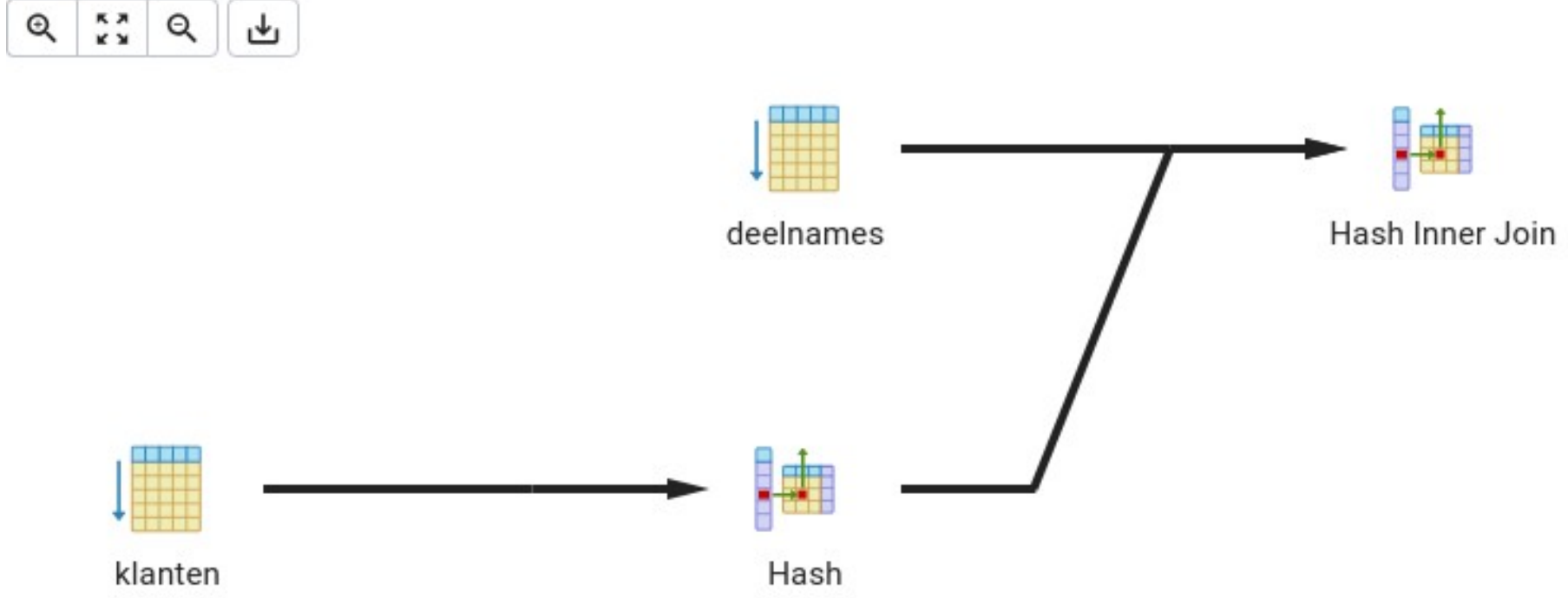
inleidende voorbeelden query plan

wim.bertels@ucll.be

```
3  select  *
4  from    klanten natural inner join deelnames;
```

Data Output | Messages | Explain | Notifications

Graphical | Analysis | Statistics



```
3  select  *
4  from    klanten natural inner join deelnames;
```

Data Output | Messages | Explain | Notifications

Graphical | Analysis | Statistics

	#	Node
	1.	→ Hash Inner Join **Hash Cond**: (deelnames.klantnr = klanten.klantnr)
	2.	→ Seq Scan on deelnames as deelnames
	3.	→ Hash
	4.	→ Seq Scan on klanten as klanten

Graphical | Analysis | Statistics

Statistics per Node Type

Node type	Count
Hash	1
Hash Inner Join	1
Seq Scan	2

Statistics per Relation

Relation name	Scan count
Node type	Count
deelnames	1
Seq Scan	1
klanten	1
Seq Scan	1

1 tabel

```
CREATE TABLE een_miljoen
    (teller     integer,
    random_tekst    text);

INSERT INTO een_miljoen
  SELECT i, md5(random()::text)
  FROM generate_series(1, 1000000) AS i;
```

```
EXPLAIN
SELECT  *
FROM een_miljoen;
```

QUERY PLAN

```
Seq Scan on een_miljoen
(cost=0.00..18918.18
rows=1058418 width=36)
```

```
ANALYZE een_miljoen;

EXPLAIN
SELECT  *
FROM    een_miljoen;
```

QUERY PLAN
Seq Scan on een_miljoen (cost=0.00..18334.00 rows=1000000 width=37)

```
EXPLAIN ANALYZE
SELECT  *
FROM    een_miljoen;
```

QUERY PLAN
Seq Scan on een_miljoen (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.008..78.234 rows=1000000 loops=1)
Planning Time: 0.012 ms
Execution Time: 106.864 ms

```
EXPLAIN
SELECT  *
FROM    een_miljoen
WHERE   teller > 500;
```

QUERY PLAN
Seq Scan on een_miljoen (cost=0.00..20834.00 rows=999507 width=37)
Filter: (teller > 500)

```
CREATE INDEX ON een_miljoen(teller);
EXPLAIN
SELECT  *
FROM    een_miljoen
WHERE   teller > 500;
```

QUERY PLAN
Seq Scan on een_miljoen (cost=0.00..20834.00 rows=999491 width=37)
Filter: (teller > 500)

```
EXPLAIN
SELECT  *
FROM    een_miljoen
WHERE   teller < 500;
```

QUERY PLAN

```
Index Scan using een_miljoen_teller_idx on een_miljoen
(cost=0.42..25.32 rows=508 width=37)
```

```
  Index Cond: (teller < 500)
```

???

filter (> 500) Seq Scan: geen index

filter (< 500) Index Scan

- Misschien..

```
SET enable_seqscan TO off;
EXPLAIN ANALYZE
SELECT  *
FROM    een_miljoen
WHERE   teller > 500;
```

QUERY PLAN
Index Scan using een_miljoen_teller_idx on een_miljoen (cost=0.42..36800.52 rows=999491 width=37) (actual time=0.023..255.981 rows=999500 loops=1)
Index Cond: (teller > 500)
Planning Time: 0.051 ms
Execution Time: 296.085 ms

QUERY PLAN
Seq Scan on een_miljoen (cost=0.00..20834.00 rows=999491 width=37)
Filter: (teller > 500)

```
SET enable_seqscan TO on;
```

meerdere tabellen

```
CREATE TABLE half_miljoen
    (teller          integer,
     random_munt     boolean);
INSERT INTO half_miljoen
    SELECT i, i%2=1
    FROM generate_series(1, 500000) AS i;
ANALYZE half_miljoen;
```

```
CREATE TABLE een_miljoen
    (teller     integer,
    random_tekst    text);

INSERT INTO een_miljoen
  SELECT i, md5(random()::text)
  FROM generate_series(1, 1000000) AS i;
```

```
EXPLAIN ANALYZE
SELECT  *
FROM    een_miljoen JOIN half_miljoen
    ON (een_miljoen.teller=half_miljoen.teller);
```

QUERY PLAN
Hash Join (cost=15417.00..60081.00 rows=500000 width=42) (actual time=102.561..679.936 rows=500000 loops=1)
Hash Cond: (een_miljoen.teller = half_miljoen.teller)
-> Seq Scan on een_miljoen (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.006..76.141 rows=1000000 loops=1)
-> Hash (cost=7213.00..7213.00 rows=500000 width=5) (actual time=102.465..102.466 rows=500000 loops=1)
Buckets: 262144 Batches: 4 Memory Usage: 6562kB
-> Seq Scan on half_miljoen (cost=0.00..7213.00 rows=500000 width=5) (actual time=0.005..33.529 rows=500000 loops=1)
Planning Time: 0.193 ms
Execution Time: 693.629 ms

```
EXPLAIN ANALYZE
SELECT  *
FROM    een_miljoen JOIN half_miljoen
    ON (een_miljoen.teller=half_miljoen.teller);
```

```
CREATE INDEX ON half_miljoen(teller);
EXPLAIN ANALYZE
SELECT  *
FROM    een_miljoen JOIN half_miljoen
    ON (een_miljoen.teller=half_miljoen.teller);
```

```
                                                    QUERY PLAN
Merge Join  (cost=1.31..40111.03 rows=500000 width=42)
            (actual time=0.014..298.152 rows=500000 loops=1)
  Merge Cond: (een_miljoen.teller = half_miljoen.teller)
  ->  Index Scan using een_miljoen_teller_idx on een_miljoen
        (cost=0.42..34317.43 rows=1000000 width=37)
        (actual time=0.006..77.998 rows=500001 loops=1)
  ->  Index Scan using half_miljoen_teller_idx on half_miljoen
        (cost=0.42..15212.42 rows=500000 width=5)
        (actual time=0.004..83.697 rows=500000 loops=1)
Planning Time: 0.264 ms
Execution Time: 311.930 ms
```

```
Hash Join   (cost=15417.00..60081.00
```

Aggregaties

```
EXPLAIN
SELECT  count(*)
FROM    een_miljoen;
```

QUERY PLAN
Finalize Aggregate (cost=14542.55..14542.56 rows=1 width=8)
-> Gather (cost=14542.33..14542.54 rows=2 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=13542.33..13542.34 rows=1 width=8)
-> Parallel Seq Scan on een_miljoen (cost=0.00..12500.67 rows=416667 width=0)

```
EXPLAIN ANALYZE
SELECT  max(random_tekst)
FROM    een_miljoen;
```

QUERY PLAN

```
Finalize Aggregate  (cost=14542.55..14542.56 rows=1 width=32)
                    (actual time=75.277..79.133 rows=1 loops=1)

  ->  Gather    (cost=14542.33..14542.54 rows=2 width=32)
                (actual time=75.198..79.124 rows=3 loops=1)

        Workers Planned: 2

        Workers Launched: 2

        ->  Partial Aggregate   (cost=13542.33..13542.34 rows=1 width=32)
                                (actual time=72.948..72.948 rows=1 loops=3)

              ->  Parallel Seq Scan on een_miljoen
                    (cost=0.00..12500.67 rows=416667 width=33)
                    (actual time=0.008..24.153 rows=333333 loops=3)

Planning Time: 0.073 ms

Execution Time: 79.155 ms
```

```
CREATE INDEX ON een_miljoen(random_tekst);
EXPLAIN ANALYZE
SELECT  max(random_tekst)
FROM    een_miljoen;
```

QUERY PLAN

```
Result  (cost=0.47..0.48 rows=1 width=32)
        (actual time=0.035..0.035 rows=1 loops=1)

  InitPlan 1 (returns $0)

    ->  Limit   (cost=0.42..0.47 rows=1 width=33)
                (actual time=0.031..0.032 rows=1 loops=1)

          ->  Index Only Scan Backward using een_miljoen_random_tekst_idx on een_miljoen
                (cost=0.42..46340.43 rows=1000000 width=33)
                (actual time=0.030..0.031 rows=1 loops=1)

                Index Cond: (random_tekst IS NOT NULL)

                Heap Fetches: 0

Planning Time: 0.156 ms

Execution Time: 0.049 ms
```

Groeperingen

```
DROP INDEX een_miljoen_random_tekst_idx;
EXPLAIN ANALYZE
SELECT  random_tekst, count(*)
FROM    een_miljoen
GROUP BY random_tekst;
```

```
                                                    QUERY PLAN
HashAggregate   (cost=105834.00..131459.00 rows=1000000 width=41)
                (actual time=473.085..1013.928 rows=1000000 loops=1)
  Group Key: random_tekst
  Planned Partitions: 16  Batches: 81  Memory Usage: 8337kB  Disk Usage: 63536kB
  ->  Seq Scan on een_miljoen
            (cost=0.00..18334.00 rows=1000000 width=33)
            (actual time=0.007..63.300 rows=1000000 loops=1)
```

```
CREATE INDEX ON een_miljoen(random_tekst);
EXPLAIN ANALYZE
SELECT  random_tekst, count(*)
FROM    een_miljoen GROUP BY random_tekst;
```

QUERY PLAN
GroupAggregate (cost=0.42..58840.43 rows=1000000 width=41) (actual time=0.030..440.101 rows=1000000 loops=1)
Group Key: random_tekst
-> Index Only Scan using een_miljoen_random_tekst_idx on een_miljoen (cost=0.42..43840.43 rows=1000000 width=33) (actual time=0.024..133.736 rows=1000000 loops=1)

Conclusie

- EXPLAIN (afhankelijk van de statistieken)

  Oplossing: ANALYZE voordien

- EXPLAIN ANALYZE

  Actuele Uitvoertijd, maar voert dus ook effectief uit!

- EXPLAIN

  Werkt ook voor andere DML (insert, ..)

  Kijk in eerste instantie naar de totale kost

  Kijk eventueel naar de scan methoden

Referenties

- Understanding Explain, guillaume.lelarge@dalibo.com