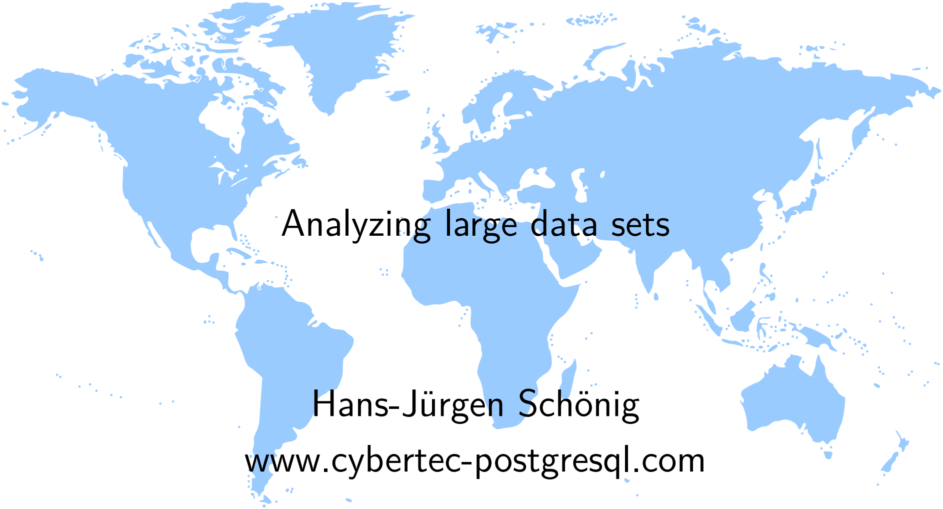

Analyzing large data sets

Hans-Jürgen Schönig

www.cybertec-postgresql.com

Who we are

Cybertec

- We are a PostgreSQL support company
- Clients around the globe
- What we do:
  - PostgreSQL 24x7 support
  - Training
  - Consulting
  - Geodata
  - Scaling
- Local office here in Tallinn

Inspiration

Ideas and failures

- This content is inspired by a database I have seen earlier this week
- Massive drama:
  - 340 billion rows
  - Oracle reached its limit
  - People tried to solve things with hardware
  - They will fail (after spending cash on Exadata) as data will grow

What they did

- A guideline to failure:
  - Joining up to 14 tables
  - No pre-aggregation
  - No thoughts on what to query how

What you should learn

- Small data sets:
  - Do basically what you want
  - Hardware is gonna bail you out
- Large data sets:
  - Stupid queries are gonna kill your
  - The more data you have, the more you have to think
- There is no “magic parameter”

Potential solutions

Aggregates and joins (1)

```
test=# CREATE TABLE t_gender (id int, name text);
CREATE TABLE
test=# INSERT INTO t_gender
    VALUES (1, 'male'), (2, 'female');
INSERT 0 2
```

Aggregates and joins (2)

```
test=# CREATE TABLE t_person (
    id       serial,
    gender   int,
    data     char(40)
);
CREATE TABLE
```

Aggregates and joins (3)

```
test=# INSERT INTO t_person (gender, data)
    SELECT  x % 2 + 1, 'data'
    FROM    generate_series(1, 5000000) AS x;
INSERT 0 5000000
```

Simple analysis

```
test=# SELECT name, count(*)
    FROM    t_gender AS a, t_person AS b
    WHERE   a.id = b.gender
    GROUP BY 1;
  name   |  count
---------+----------
 female | 2500000
 male   | 2500000
(2 rows)
Time: 961.034 ms
```

Can we speed it up?

- Does anybody see a way to make this faster?
- The answer is “deep” inside the planner

Let us try this one

```
test=# WITH x AS
(
        SELECT gender, count(*) AS res
        FROM    t_person AS a
        GROUP BY 1
)
SELECT  name, res
FROM    x, t_gender AS y
WHERE x.gender = y.id;
... <same result> ...
Time: 526.472 ms
```

How did it happen?

- We do not understand . . .
- It must be a miracle ;)

Understanding the planner (1)

- The answer is deep inside the planner
- Let us see what happens if we use just one CPU core:

```
test=# SET max_parallel_workers_per_gather TO 0;
SET
```

Understanding the planner (2)

```
explain SELECT name, count(*)
    FROM    t_gender AS a, t_person AS b
    WHERE a.id = b.gender GROUP BY 1;
             QUERY PLAN
---------------------------------
 HashAggregate ...
  Group Key: a.name
  ->  Hash Join  (rows=5000034)
      Hash Cond: (b.gender = a.id)
      ->  Seq Scan on t_person b  (rows=5000034)
      ->  Hash  (cost=1.02..1.02 rows=2 width=10)
          ->  Seq Scan on t_gender a  (rows=2)
```

Understanding the planner (3)

- The join is performed BEFORE the aggregation
    - Millions of lookups
- This causes the change in performance

Understanding the planner (4)

```
test=# explain WITH x AS
(
        SELECT gender, count(*) AS res
        FROM    t_person AS a
        GROUP BY 1
)
SELECT  name, res
FROM    x, t_gender AS y
WHERE x.gender = y.id;
```

Understanding the planner (5)

```
          QUERY PLAN
---------------------------------
 Hash Join  (rows=2)
  Hash Cond: (y.id = x.gender)
  CTE x
  ->  HashAggregate  (rows=2)
      Group Key: a.gender
      ->  Seq Scan on t_person a
        (rows=5000034)
  ->  Seq Scan on t_gender y  (rows=2)
  ->  Hash  (rows=2)
      ->  CTE Scan on x  (rows=2)
```

Lessons learned

- Difference is irrelent if your amount of data is very small
- Small things can make a difference
- Good news: An in-core fix is on the way for (maybe) PostgreSQL 11.0.
  - If you are REALLY in need, we can help

One more classical example

- Processing A LOT of data
  - Suppose we have 20 years worth of data
  - 1 billion rows per year

```
SELECT  sensor, count(temp)
FROM    t_sensor
WHERE   t BETWEEN '2014-01-01'
    AND '2014-12-31'
GROUP BY sensor;
```

Observations

- Reading 1 billion out of 20 billion rows can be slow
- A classical btree might be a nightmare too
  - A lot of random I/O
  - Size is a round 20.000.000.000 * 25 bytes
- We can do A LOT better

Ideas

- Partition data by year
  - A sequential scan on 1 billion rows is A LOT better than using a btree
  - The planner will automatically kick out unnecessary partitions
- Alternatively:
  - Use brin indexes (Block range indexes)

An example (1)

```
test=# CREATE INDEX idx_btree ON t_person (id);
CREATE INDEX
Time: 1542.177 ms (00:01.542)

test=# CREATE INDEX idx_brin ON t_person USING brin(id);
CREATE INDEX
Time: 721.838 ms
```

An example (2)

```
test=# \di+
            List of relations
   Name    | Type  |   Table   |  Size
-----------+-------+-----------+-------
 idx_brin  | index |  t_person | 48 kB
 idx_btree | index |  t_person | 107 MB
(2 rows)
```

BRIN at work

- Takes 128 blocks
  - Stores min + max value of the block
- Super small (2000 x smaller than btrees)
- Only works well when there is correlation

Doing many things at once

Passing over data too often

- One source of trouble is to read data too often
- Some ideas:
  - Use grouping sets and partial aggregates
  - Use synchronous sequential scans
  - Use pre-aggregation

Grouping sets: Doing more at once

- Preparing some data

```
test=# ALTER TABLE t_person
    ADD COLUMN age int DEFAULT random()*100;
ALTER TABLE

test=# SELECT * FROM t_person LIMIT 4;
   id    | gender |   data    | age
---------+--------+-----------+-----
 5000001 |      2 | data      |  78
 5000002 |      1 | data      |  26
 5000003 |      2 | data      |  33
 5000004 |      1 | data      |  55
```

Adding ROLLUP

```
test=# SELECT   name, count(*)
FROM    t_gender AS a, t_person AS b
WHERE   a.id = b.gender
GROUP BY ROLLUP(1)
ORDER BY 1;
  name  |  count
--------+---------
 female | 2500000
 male   | 2500000
        | 5000000
```

Adding partial aggregates

```
test=# SELECT   name,
       count(*) AS everybody,
       count(*) FILTER (WHERE age < 50) AS young,
       count(*) FILTER (WHERE age >= 50) AS censored
FROM    t_gender AS a, t_person AS b
WHERE   a.id = b.gender
GROUP BY ROLLUP(1)
ORDER BY 1;
  name  | everybody |  young  | censored
--------+-----------+---------+----------
 female |   2500000 | 1238156 |  1261844
 male   |   2500000 | 1238403 |  1261597
        |   5000000 | 2476559 |  2523441
```

Wrapping things up

- Remember:
  - Those are just some ideas
  - You can do a lot more
- A lot of what I sad is true for PostgreSQL AND Oracle, etc.
  - You can still get rid of Oracle ;)

Finally

- Thank you for your attention
- Any questions?

Contact us

Cybertec Schönig & Schönig GmbH
Hans-Jürgen Schönig
Gröhrmühlgasse 26
A-2700 Wiener Neustadt

Email: hs@cybertec.at
www.postgresql-support.de

Follow us on Twitter: @PostgresSupport