1.– 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Schemafrei

PostgeSQL: Die NoSQL-Datenbank, die keiner kennt

Stephan Hochdörfer

bitExpert AG

bitExpert

SOFTWARE DEVELOPMENT | MEDIA TECHNOLOGY

PostgreSQL: Die NoSQL Datenbank

Stephan Hochdörfer // 03.09.2014

Über mich

- Stephan Hochdörfer
- Head of Technology, bitExpert AG (Mannheim)
- S.Hochdoerfer@bitExpert.de
- @shochdoerfer

Ende des relationalen Wegs?

"Freiheit" durch NoSQL?

NoSQL Datenbanken

NoSQL Klassifizierung

- Graphdatenbanken
- Objektdatenbanken
- Spaltenorientierte Datenbanken
- Key-Value Datenbanken
- Dokumentenorientierte Datenbanken

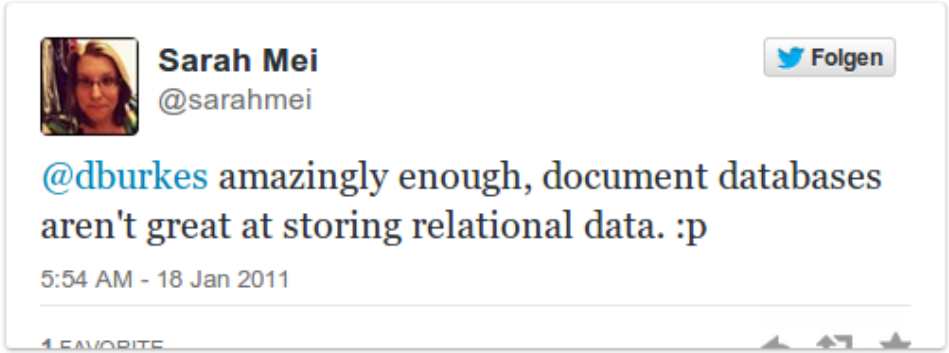
Schema vs. Schemafrei

- Schemaänderungen können teuer sein
- Schemafreiheit beschleunigt Entwicklung
- Schemafreiheit begünstigt Continuous Delivery
- Schemafrei heißt nicht unstrukturiert

Friedliche Co-Existenz...

- Speichere relationale Daten relational
- Speichere Dokumente dokumentenorientiert

...aber richtig entscheiden!



PostgreSQL als Bindeglied

Postgres Datentypen

"Schemafreie" Datentypen:

- Array
- hstore
- JSON / JSONb
- XML

Array

« [...] PostgreSQL allows columns of a table to be defined as variable-length multidimensional arrays. »
- PostgreSQL documentation

Array Spalte anlegen

```
CREATE TABLE blogposts (
    title    text,
    content  text,
    tags     text[]
);
```

Multidimensionales Array

```
CREATE TABLE tictactoe (
  squares   integer[3][3]
);
```

Daten als Array speichern

```
INSERT INTO blogposts VALUES (
    'Mein Blogpost',
    'Lorem ipsum...',
    ARRAY['lorem', 'ipsum', 'blogpost']
);

INSERT INTO blogposts VALUES (
    'Mein zweiter Blogpost',
    'Lorem ipsum...',
    ARRAY['lorem', 'ipsum']
);
```

Daten als Array speichern

```
INSERT INTO blogposts VALUES (
    'Mein Blogpost',
    'Lorem ipsum...',
    '{"lorem", "ipsum", "blogpost"}'
);

INSERT INTO blogposts VALUES (
    'Mein zweiter Blogpost',
    'Lorem ipsum...',
    '{"lorem", "ipsum"}'
);
```

Array Elemente abfragen

```
SELECT tags[3] FROM blogposts;
```

```
   tags
----------
 blogpost

(2 rows)
```

Arrays durchsuchen

```
SELECT * from blogposts WHERE tags[2] = 'ipsum';
```

```
         title          |    content     |          tags
------------------------+----------------+------------------------
 Mein Blogpost          | Lorem ipsum... | {lorem,ipsum,blogpost}
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
(2 rows)
```

Arrays durchsuchen

```
SELECT * FROM blogposts WHERE 'blogpost' = ANY(tags);
```

```
     title      |    content      |          tags
----------------+-----------------+------------------------
 Mein Blogpost | Lorem ipsum... | {lorem,ipsum,blogpost}
(1 row)
```

Arrays durchsuchen

```
SELECT * FROM blogposts WHERE tags && ARRAY['lorem', 'blogpost'];
```

```
         title          |    content     |          tags
------------------------+----------------+------------------------
 Mein Blogpost          | Lorem ipsum... | {lorem,ipsum,blogpost}
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
(2 rows)
```

Arrays durchsuchen

```
SELECT * FROM blogposts WHERE tags @> ARRAY['lorem', 'blogpost']
```

```
     title      |    content      |          tags
----------------+-----------------+------------------------
 Mein Blogpost | Lorem ipsum... | {lorem,ipsum,blogpost}
(1 row)
```

Tagcloud ermitteln

```
SELECT UNNEST(tags) as tag, COUNT(*) as cnt FROM blogposts
GROUP BY tag;
```

```
   tag     | cnt
-----------+-----
 blogpost |   1
 lorem    |   2
 ipsum    |   2
(3 rows)
```

Arrays indexieren

```
CREATE INDEX tags_idx on "blogposts" USING GIN ("tags");
```

```
SET enable_seqscan TO off;

EXPLAIN ANALYZE SELECT * FROM blogposts WHERE tags @> ARRAY['tag3'];
```

```
                                QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on blogposts  (cost=8.00..12.01 rows=1 width=96) (actual time=0.031..
   Recheck Cond: (tags @> '{tag3}'::text[])
   ->  Bitmap Index Scan on tags_idx  (cost=0.00..8.00 rows=1 width=0) (actual time=0.0
         Index Cond: (tags @> '{tag3}'::text[])
 Total runtime: 0.131 ms
(5 rows)
```

Arrays modifizieren

```
UPDATE blogposts  SET tags = ARRAY['tag1', 'tag2', 'blogpost']
  WHERE title = 'Mein Blogpost';
```

```
SELECT * FROM blogposts;
```

```
         title          |    content     |          tags
------------------------+----------------+-----------------------
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
 Mein Blogpost         | Lorem ipsum... | {tag1,tag2,blogpost}
(2 rows)
```

Arrays modifizieren

```
UPDATE blogposts SET tags[3] = 'tag3' WHERE title = 'Mein Blogpost';
```

```
SELECT * FROM blogposts;
```

```
         title          |    content     |       tags
------------------------+----------------+------------------
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
 Mein Blogpost         | Lorem ipsum... | {tag1,tag2,tag3}
(2 rows)
```

Arrays modifizieren

```
UPDATE blogposts SET tags[5] = 'tag6' WHERE title = 'Mein Blogpost';
```

```
SELECT * FROM blogposts;
```

```
         title          |    content     |             tags
------------------------+----------------+------------------------------
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
 Mein Blogpost         | Lorem ipsum... | {tag1,tag2,tag3,NULL,tag6}
(2 rows)
```

Arrays modifizieren

```
UPDATE blogposts SET tags[4:5] = ARRAY['tag4', 'tag5']
WHERE title = 'Mein Blogpost';
```

```
         title          |    content     |              tags
------------------------+----------------+--------------------------------
 Mein zweiter Blogpost | Lorem ipsum... | {lorem,ipsum}
 Mein Blogpost         | Lorem ipsum... | {tag1,tag2,tag3,tag4,tag5}
(2 rows)
```

Array Funktionen

- array_dims()
- array_upper()
- array_lower()
- array_length()
- array_prepend()
- array_append()
- ...

hstore

« [...] data type for storing sets of key/value pairs within a single PostgreSQL value. [...] Keys and values are simply text strings. » - PostgreSQL documentation

hstore Extension installieren

```
CREATE EXTENSION hstore;
```

hstore Spalte anlegen

```
CREATE TABLE products (
    name          text,
    price         float,
    description   text,
    metadata      hstore
);
```

hstore Daten speichern

```
INSERT INTO products VALUES (
    'Product A',
    123.3,
    'Lorem ipsum',
    'color => black, weight => 3kg'::hstore
);


INSERT INTO products VALUES (
    'Product B',
    45.0,
    'Lorem ipsum',
    'color => red'::hstore
);
```

hstore Keys abfragen

```
SELECT metadata->'color' FROM products;
```

```
 ?column?
----------
 black
 red
(2 rows)
```

hstore Keys abfragen

```
SELECT metadata->'weight' FROM products;
```

```
 ?column?
----------
 3kg

(2 rows)
```

hstore Keys suchen

```
SELECT * FROM products WHERE metadata ? 'weight';
```

```
   name    | price | description |             metadata
-----------+-------+-------------+----------------------------------
 Product A | 123.3 | Lorem ipsum | "color"=>"black", "weight"=>"3kg"
(1 row)
```

hstore Key/Value suchen

```
SELECT * FROM products WHERE metadata @> 'color => red';
```

```
   name     | price | description |    metadata
------------+-------+-------------+----------------
 Product B |    45 | Lorem ipsum | "color"=>"red"
(1 row)
```

hstore Keys ermitteln

```
SELECT DISTINCT UNNEST(AKEYS(metadata)) as keys FROM products;
```

```
  keys
--------
 weight
 color
(2 rows)
```

hstore modifizieren

```
UPDATE products SET metadata = 'color => black, weight => 5kg'::hstore WHERE name = 'Pr
```

hstore modifizieren

```
UPDATE products SET metadata = metadata || 'weight => 1kg'::hstore WHERE name = 'Produc
```

hstore modifizieren

```
UPDATE products SET metadata = metadata - 'weight'::text;
```

hstore indexieren

```
CREATE INDEX metadata_idx ON products USING GIN (metadata);
```

```
EXPLAIN ANALYZE SELECT * FROM products WHERE metadata @>
'color => red';
```

```
                                   QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on products  (cost=12.00..16.01 rows=1 width=104) (actual time=0.044.
   Recheck Cond: (metadata @> '"color"=>"red"'::hstore)
   ->  Bitmap Index Scan on metadata_idx  (cost=0.00..12.00 rows=1 width=0) (actual tim
         Index Cond: (metadata @> '"color"=>"red"'::hstore)
 Total runtime: 0.100 ms
(5 rows)
```

hstore Funktionen

- akeys()
- avals()
- hstore_to_array()
- hstore_to_json()
- ...

hstore Beispiel: Auditing

```
CREATE TABLE audit
(
    change_date timestamptz default now(),
    before                          hstore,
    after                           hstore
);
```

```
create function audit()
  returns trigger
  language plpgsql
as $$
begin
  INSERT INTO audit(before, after)
       SELECT hstore(old), hstore(new);
  return new;
end;
$$;
```

hstore Beispiel: Auditing

```
create trigger audit
      after update on example
          for each row
execute procedure audit();
```

```
select change_date, after - before as diff from audit;

          change_date           |   diff
--------------------------------+-----------
 2013-11-08 09:29:19.808217+02 | "f1"=>"b"
 2013-11-08 09:29:19.808217+02 | "f2"=>"c"

(2 rows)
```

Quelle: Auditing Changes with Hstore

JSON

« [...] the json data type has the advantage of checking that each stored value is a valid JSON value. »
- PostgreSQL documentation

JSON Spalte anlegen

```
CREATE TABLE products (
    name          text,
    price         float,
    description   text,
    metadata      json
);
```

JSON Daten speichern

```
INSERT INTO products VALUES (
    'Product A',
    123.3,
    'Lorem ipsum',
    ' { "color": "black", "weight": "3kg" } '
);


INSERT INTO products VALUES (
    'Product B',
    45.0,
    'Lorem ipsum',
    ' { "color": "red" } '
);
```

JSON Daten speichern

```
INSERT INTO products VALUES (
    'Product C',
    9.95,
    'Lorem ipsum',
    ' { "color": ["red", "green", "blue"] } '
);
```

JSON Daten speichern

```
INSERT INTO products VALUES (
    'Product D',
    9.95,
    'Lorem ipsum',
    ' { "color": ["red", "green", "blue", ] } '
);
```

```
ERROR:  invalid input syntax for type json
LINE 5:     ' { "color": ["red", "green", "blue", ] } '
            ^
DETAIL:  Expected JSON value, but found "]".
CONTEXT:  JSON data, line 1:  { "color": ["red", "green", "blue", ]...
```

JSON Daten abfragen

```
SELECT metadata->>'color' FROM products;
```

```
         ?column?
--------------------------
 black
 red
 ["red", "green", "blue"]
(3 rows)
```

JSON Daten abfragen

```
SELECT metadata#>'{color, 2}' FROM products;
```

```
 ?column?
----------


 "blue"
(3 rows)
```

JSON Daten modifizieren

```
UPDATE products SET metadata = ' { "color": ["red", "green", "blue", "yellow"] } ' WHER
```

JSON Inhalte indexieren

```
CREATE UNIQUE INDEX product_color ON products((metadata->>'color'));
```

```
INSERT INTO products VALUES (
    'Product D',
    21.95,
    'Lorem ipsum',
    ' { "color": "red" } '
);
```

```
ERROR:  duplicate key value violates unique constraint "product_color"
DETAIL:  Key ((metadata ->> 'color'::text))=(red) already exists.
```

JSON Funktionen

- array_to_json()
- row_to_json()
- json_array_length()
- json_object_keys()
- ...

Micro-JSON Webservice

```
SELECT row_to_json(products) FROM products WHERE name = 'Product C';
```

```
                                row_to_json
-------------------------------------------------------------------------
 {"name":"Product C","price":9.95,"description":"Lorem ipsum","metadata": { "color": ["
(1 row)
```

Micro-JSON Webservice

```
SELECT row_to_json(t)
FROM (
  SELECT name, price FROM products WHERE name = 'Product C'
) t;
```

```
            row_to_json
-----------------------------------
 {"name":"Product C","price":9.95}
(1 row)
```

JSONb

« There are two JSON data types: json and jsonb [...]
The major practical difference is one of efficiency. »
- PostgreSQL documentation

JSONb Spalte anlegen

```
CREATE TABLE products (
    name          text,
    price         float,
    description   text,
    metadata      jsonb
);
```

JSONb Daten speichern

```
INSERT INTO products VALUES (
    'Product A',
    123.3,
    'Lorem ipsum',
    ' { "color": "black", "weight": "3kg" } '
);


INSERT INTO products VALUES (
    'Product B',
    45.0,
    'Lorem ipsum',
    ' { "color": "red" } '
);
```

JSONb Spalte indexieren

```
CREATE INDEX metadata_idx ON products USING gin (metadata);
```

JSONb Spalte durchsuchen

```
SELECT * FROM products WHERE metadata @> '{"color": "black"}';
```

```
    name     |  price  |   description    |          metadata
-------------+---------+------------------+----------------------------
 Product A   |  123.3  |   Lorem ipsum    | {"color": "black", "weight": "3kg"}

(1 row)
```

JSONb Spalte durchsuchen

```
EXPLAIN ANALYZE SELECT * FROM products WHERE metadata @> '{"color": "black"}';
```

```
                                QUERY PLAN
-----------------------------------------------------------------------------
 Bitmap Heap Scan on products  (cost=12.00..16.01 rows=1 width=104) (actual time=0.048.
   Recheck Cond: (metadata @> '{"color": "black"}'::jsonb)
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on metadata_idx  (cost=0.00..12.00 rows=1 width=0) (actual tim
         Index Cond: (metadata @> '{"color": "black"}'::jsonb)
 Planning time: 0.131 ms
 Execution time: 0.105 ms
```

JSONb Keys suchen

```
SELECT * FROM products WHERE metadata ? 'weight';
```

```
    name     |  price  |   description    |          metadata
-------------+---------+------------------+----------------------------
 Product A   |  123.3  |   Lorem ipsum    | {"color": "black", "weight": "3kg"}

(1 row)
```

JSONb Keys suchen

```
SELECT * FROM products WHERE metadata ?| array['weight', 'color'];
```

```
    name     |  price  |   description   |          metadata
-------------+---------+-----------------+----------------------------
 Product A   |  123.3  |   Lorem ipsum   | {"color": "black", "weight": "3kg"}
 Product B   |  45     |   Lorem ipsum   | {"color": "red"}
```

JSONb Keys suchen

```
SELECT * FROM products WHERE metadata ?& array['weight', 'color'];
```

```
    name     |  price  |   description    |          metadata
-------------+---------+------------------+----------------------------
 Product A   |  123.3  |   Lorem ipsum    | {"color": "black", "weight": "3kg"}

(1 row)
```

JSON vs. JSONb

- JSONb ideal für Sortierungen, Indexe, Suchabfragen etc.
- JSONb benötigt i.d.R. mehr Speicherplatz
- JSON verwenden wenn Daten nur gespeichert werden
- JSON verwenden wenn Struktur wichtig ist!

XML

« [...] it checks the input values for well-formedness, and there are support functions to perform type-safe operations on it. » - PostgreSQL documentation

XML Spalte anlegen

```
CREATE TABLE imports (
    importDate date,
    content    XML
);
```

XML Daten speichern

```
INSERT INTO imports VALUES (
    '2014-04-05',
    '<?xml version="1.0"?>
      <order customer="4711">
        <product id="1234">10</product>
        <product id="9876">50</product>
      </order>'::xml
);

INSERT INTO imports VALUES (
    '2014-04-05',
    '<?xml version="1.0"?>
      <order customer="5432">
        <product id="1234">3</product>
    </order>'::xml
);
```

XML Daten speichern

```
INSERT INTO imports VALUES (
    '2014-04-05',
    '<?xml version="1.0"?>
      <open></close>'::xml
);
```

```
ERROR:  invalid XML content
LINE 3:     '<?xml version="1.0"?>
            ^
DETAIL:  line 2: Opening and ending tag mismatch: open line 2 and close
      <open></close>
                    ^
line 2: chunk is not well balanced
      <open></close>
                    ^
```

XML Daten abfragen

```
SELECT xpath('/order/product/@id', content) FROM imports;
```

```
    xpath
-------------
 {1234,9876}
 {1234}
(2 rows)
```

XML Daten abfragen

```
SELECT UNNEST(xpath('/order/product/@id', content)::text[])
as productid, COUNT(*) as orders FROM imports GROUP BY productid;
```

```
 productid | orders
-----------+--------
 9876      |      1
 1234      |      2
(2 rows)
```

XML Daten durchsuchen

```
SELECT * FROM imports WHERE xpath_exists('/order/product[@id=9876]',  content);
```

```
 importdate |              content
------------+-----------------------------------
 2014-04-05 |       <order customer="4711">  +
            | <product id="1234">10</product>+
            | <product id="9876">50</product>+
            |       </order>
(1 row)
```

XML Export erzeugen

```
SELECT query_to_xml('SELECT * FROM products', true, false, '') as product;
```

```
                                product
-----------------------------------------------------------------------
 <table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">      +
                                                                    +
 <row>                                                              +
   <name>Product A</name>                                           +
   <price>123.3</price>                                             +
   <description>Lorem ipsum</description>                           +
   <metadata> { "color": "black", "weight": "3kg" } </metadata>     +
 </row>                                                             +
                                                                    +
 </table>                                                           +

(1 row)
```

XML Funktionen

- xmlconcat()
- xmlforest()
- xmlagg()
- xml_is_well_formed()
- query_to_xml()
- xpath()
- ...

Tabellenpartitionierung

« [...] Partitioning refers to splitting what is logically one large table into smaller physical pieces. »
- PostgreSQL documentation

Tabellenpartitionierung

```
CREATE TABLE sales_stats (
    year              int,
    product           text,
    sales_by_quarter  hstore
);
```

```
CREATE TABLE sales_stats_2012(
    CHECK ( year = 2012 )
) INHERITS (sales_stats);

CREATE TABLE sales_stats_2011(
    CHECK ( year = 2011 )
) INHERITS (sales_stats);
```

Tabellenpartitionierung

```
CREATE OR REPLACE FUNCTION sales_stats_part()
RETURNS TRIGGER AS
$BODY$
DECLARE
   _new_time int;
   _tablename text;
   _year text;
   _result record;
BEGIN
   _new_time := ((NEW."time"/86400)::int)*86400;
   _year := to_char(to_timestamp(_new_time), 'YYYY');
   _tablename := 'sales_stats_'||_year;

   PERFORM 1
   FROM   pg_catalog.pg_class c
   JOIN   pg_catalog.pg_namespace n ON n.oid = c.relnamespace
   WHERE  c.relkind = 'r' AND c.relname = _tablename;
```

Tabellenpartitionierung

```
  IF NOT FOUND THEN
    EXECUTE 'CREATE TABLE ' || quote_ident(_tablename) || ' (
      CHECK ( year = ' || quote_literal(_year) || '))
    ) INHERITS (sales_stats)';
  END IF;

  EXECUTE 'INSERT INTO ' || quote_ident(_tablename) || ' VALUES ($1.*)' USING NEW;
  RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER sales_stats_trigger
BEFORE INSERT ON sales_stats
FOR EACH ROW EXECUTE PROCEDURE sales_stats_part();
```

...und darüber hinaus

- Foreign Data Wrappers
- Conditional Indexes
- Range Datentyp
- Materialized views
- ...

Umfangreiches Ökosystem

Große Nutzerbasis

Vielen Dank für die Aufmerksamkeit!