itspeople

# CSV mit reinem SQL & der Magie von JSON_TABLE einlesen

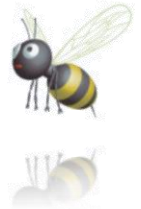
17. November 2016

**Robert Marz**

2016 DOAG Konferenz + Ausstellung

# Robert Marz

| | |
|---|---|
| Kunde | • Technical Architect<br>mit datenbankzentrischem Weltbild |
| its-people | • Portfoliomanager Datenbanken<br>• Blogredakteur |
| DOAG | • Themenverantwortlicher „Cloud“<br>in der Datenbank Community |



@RobbieDatabee

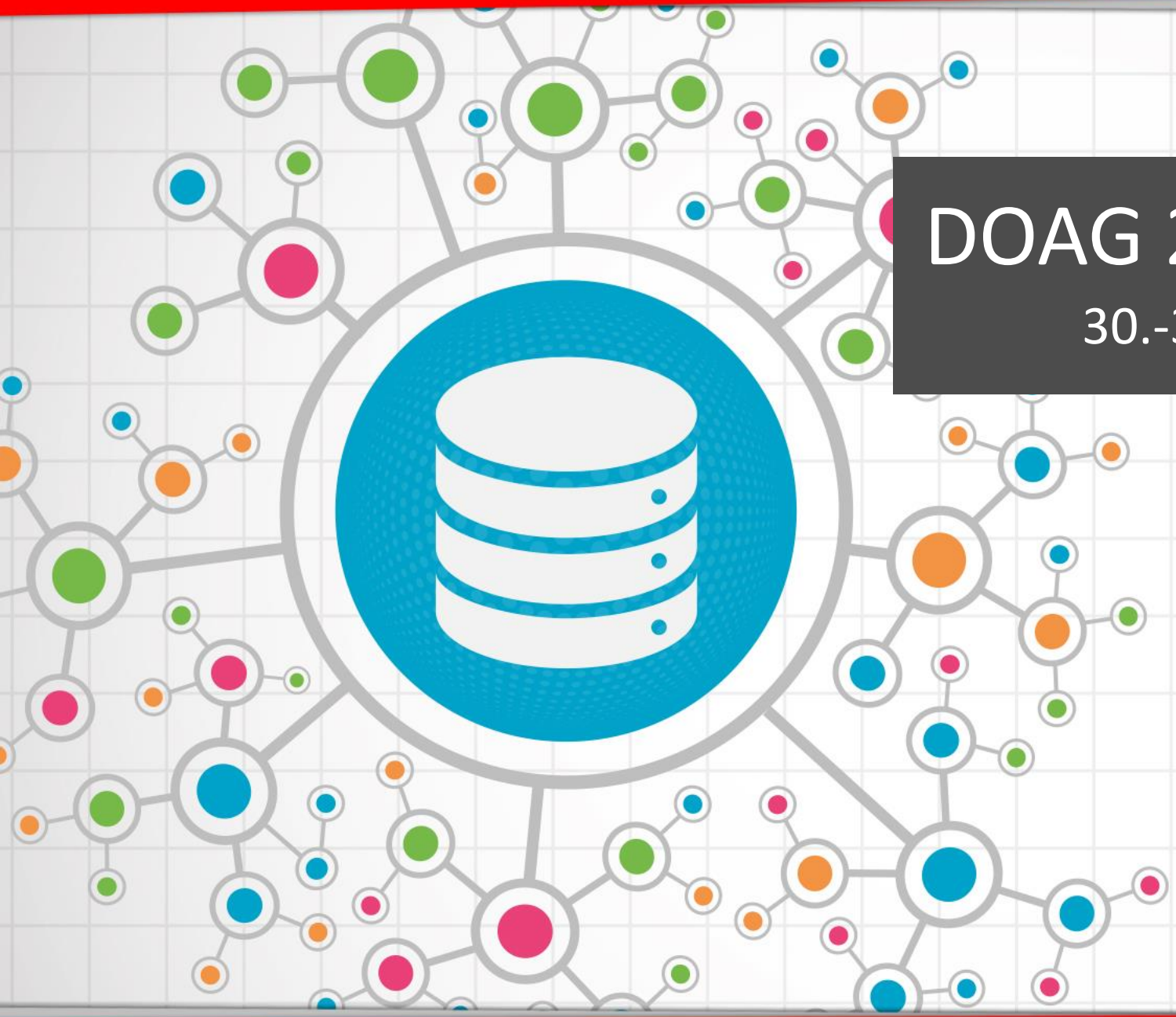
blog.its-people.de

Robert.Marz
@its-people.de

# its-people auf der DOAG 2016


- Quo vadis datum?
  Data Lineaging in gewachsenen Warehouse-Strukturen
  - Jens Behring
  - Dienstag, 15.11.  13:00 Uhr Helsinki
- **CSV mit reinem SQL und der Magie von JSON_TABLE einlesen**
  - Robert Marz
  - Dienstag, 15.11 14:00 Uhr Oslo
- Eine Karte sagt mehr als 1000 Worte
  - Sven Brömer
  - Mittwoch 16.11. 12:00 Uhr Oslo
- Scripting mit SQLcl
  Batchscripts auf einem neuen Level
  - Sabine Heimsath, Robert Marz
  - Mittwoch, 16.11. 14:00 Uhr Kopenhagen
- Panel:
  Der DBA in der Cloud
  - Moderator Robert Marz
  - Donnerstag, 17.11. 10:00 Uhr Kiew
- Werkzeuge für DBAs und Cloudnutzer: ssh
  - Robert Marz
  - Donnerstag, 17.11. 16:00 Uhr Oslo

DOAG



# DOAG 2017 **Datenbank**

30.-31. Mai 2017 in Düsseldorf

datenbank.doag.org

Ab sofort Vortrag einreichen!

# Motivation

CSV-Daten sind allgegenwärtig

PL/SQL ist umständlich

Ask Tom Frage:

parsing a CLOB field which contains CSV data

# CSV Dateien

gewachsenes Format

```
Eins,Zwei,Drei
Vier,Fünf
Sechs,Sieben,Acht
```

Ein bisschen standardisiert:

- RFC4180 für CSV

# JSON Dateien

| | |
|---|---|
| Natives Format von JavaScript | "XML mit Klammern statt Tags" |
| Schemaless | Unstrukturierte Daten |
| | Parser prüft nicht gegen DTD |
| Strukturen | {} - Objekt |
| | [] - Array |
| Zuweisung | "Variable" : "Wert" |

```
{
  "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to create
              markup languages such as DocBook.",
            "GlossSeeAlso": [
              "GML",
              "XML"
            ]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

# JSON Datentypen

| | | |
|---|---|---|
| JSON | String | "text" : "Hallo Welt" |
| | Zahl | "integer" : 12345 |
| | | "double" : 123.45 |
| | | "float" : 1.234e-6 |
| | Boolean | „Wahr" : true |
| | | „Falsch" : false |
| | | "Unbestimmt" : null |
| | RFC7159 für JSON | |

# Vergleich JSON - CSV

```
Eins,Zwei,Drei
Vier,Fünf
Sechs,Sieben,Acht
```

```
["Eins", "Zwei", "Drei"
,"Vier", "Fünf"
, "Sechs", "Sieben", "Acht"]
```

# Der JSON_TABLE Operator

JSON_TABLE

- Lebt in der SQL-From-Clause
- Produziert **Zeilen** und **Spalten**
- Akzeptiert JSON aus CLOBs
- Aufnahme in den SQL-Standard

# Der JSON_TABLE Operator

```
select wert
  from json_table( '["Eins", "Zwei", "Drei",
                     "Vier", "Fünf", "Sechs"]'
                 , '$[*]'
                 columns wert varchar2 path '$'
                 )
/
```

Das JSON-Dokument

Erzeugt Zeilen

Erzeugt Spalten

```
WERT
--------------------
Eins
Zwei
Drei
Vier
Fünf
Sechs

6 rows selected

Elapsed: 00:00:00.011
```

# Überführen von CSV nach JSON

Anforderung

erhalten der Zeilen- und Spaltenstruktur

Lösung

geschachteltes JSON-Array

```
[
  ["Eins", "Zwei", "Drei"]
, ["Vier", "Fünf"]
, [ "Sechs", "Sieben", "Acht"]
]
```

# Die REGEXP_REPLACE-Funktion

REGEXP_REPLACE

Lebt in der

- SQL-Select-List
- Where-Clause

Arbeitet wie replace()

- reguläre Ausdrücke statt statischer Strings

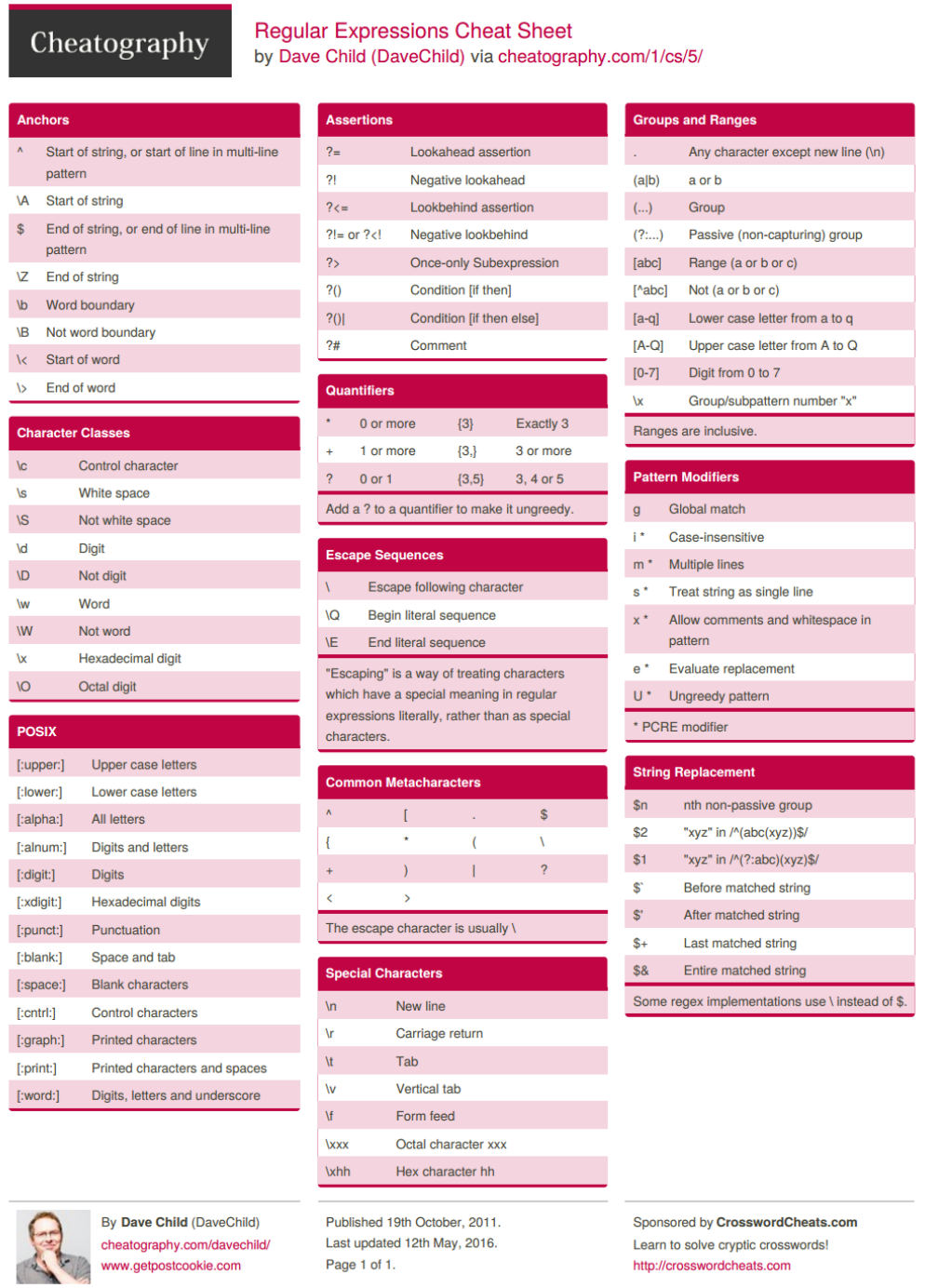
Akzeptiert CLOBs

# Reguläre Ausdrücke - Kurzüberblick

RegExp in kurz und schnell geht nicht.

Google: RegExp Tutorial|Einführung|Tipps

https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/pdf/



Cheatography

Regular Expressions Cheat Sheet
by Dave Child (DaveChild) via cheatography.com/1/cs/5/

**Anchors**

| | |
|---|---|
| ^ | Start of string, or start of line in multi-line pattern |
| \A | Start of string |
| $ | End of string, or end of line in multi-line pattern |
| \Z | End of string |
| \b | Word boundary |
| \B | Not word boundary |
| \< | Start of word |
| \> | End of word |

**Character Classes**

| | |
|---|---|
| \c | Control character |
| \s | White space |
| \S | Not white space |
| \d | Digit |
| \D | Not digit |
| \w | Word |
| \W | Not word |
| \x | Hexadecimal digit |
| \O | Octal digit |

**POSIX**

| | |
|---|---|
| [:upper:] | Upper case letters |
| [:lower:] | Lower case letters |
| [:alpha:] | All letters |
| [:alnum:] | Digits and letters |
| [:digit:] | Digits |
| [:xdigit:] | Hexadecimal digits |
| [:punct:] | Punctuation |
| [:blank:] | Space and tab |
| [:space:] | Blank characters |
| [:cntrl:] | Control characters |
| [:graph:] | Printed characters |
| [:print:] | Printed characters and spaces |
| [:word:] | Digits, letters and underscore |

**Assertions**

| | |
|---|---|
| ?= | Lookahead assertion |
| ?! | Negative lookahead |
| ?<= | Lookbehind assertion |
| ?!= or ?<! | Negative lookbehind |
| ?> | Once-only Subexpression |
| ?() | Condition [if then] |
| ?()\| | Condition [if then else] |
| ?# | Comment |

**Quantifiers**

| | | | |
|---|---|---|---|
| * | 0 or more | {3} | Exactly 3 |
| + | 1 or more | {3,} | 3 or more |
| ? | 0 or 1 | {3,5} | 3, 4 or 5 |

Add a ? to a quantifier to make it ungreedy.

**Escape Sequences**

| | |
|---|---|
| \ | Escape following character |
| \Q | Begin literal sequence |
| \E | End literal sequence |

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

**Common Metacharacters**

| | | | |
|---|---|---|---|
| ^ | [ | . | $ |
| { | * | ( | \ |
| + | ) | \| | ? |
| < | > | | |

The escape character is usually \

**Special Characters**

| | |
|---|---|
| \n | New line |
| \r | Carriage return |
| \t | Tab |
| \v | Vertical tab |
| \f | Form feed |
| \xxx | Octal character xxx |
| \xhh | Hex character hh |

**Groups and Ranges**

| | |
|---|---|
| . | Any character except new line (\n) |
| (a\|b) | a or b |
| (...) | Group |
| (?:...) | Passive (non-capturing) group |
| [abc] | Range (a or b or c) |
| [^abc] | Not (a or b or c) |
| [a-q] | Lower case letter from a to q |
| [A-Q] | Upper case letter from A to Q |
| [0-7] | Digit from 0 to 7 |
| \x | Group/subpattern number "x" |

Ranges are inclusive.

**Pattern Modifiers**

| | |
|---|---|
| g | Global match |
| i * | Case-insensitive |
| m * | Multiple lines |
| s * | Treat string as single line |
| x * | Allow comments and whitespace in pattern |
| e * | Evaluate replacement |
| U * | Ungreedy pattern |

* PCRE modifier

**String Replacement**

| | |
|---|---|
| $n | nth non-passive group |
| $2 | "xyz" in /^(abc(xyz))$/ |
| $1 | "xyz" in /^(?:abc)(xyz)$/ |
| $` | Before matched string |
| $' | After matched string |
| $+ | Last matched string |
| $& | Entire matched string |

Some regex implementations use \ instead of $.

By **Dave Child** (DaveChild)
cheatography.com/davechild/
www.getpostcookie.com

Published 19th October, 2011.
Last updated 12th May, 2016.
Page 1 of 1.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
http://crosswordcheats.com

# JSON Umwandlung: Array - Zeilenanfang

```
with
  csv as (
           select to_clob( '"eins", "zwei", "drei"'||chr(10)
                         || '"vier", "fünf", "sechs"'||chr(10)
                         || '"sieben", "acht", "neun"'||chr(10)
                          ) blb
             from dual
         )
, jsn1 as (  -- Zeilenanfang
            select regexp_replace( blb
                                 , '^'
                                 ,'['
                                 ,1,0,'m'
                                 ) blb
              from csv
         )
```

```
BLB
----------------------------------------
["eins", "zwei", "drei"
["vier", "fünf", "sechs"
["sieben", "acht", "neun"
```

# JSON Umwandlung: Array - Zeilenende

```
, jsn1 as (  -- Zeilenanfang
           select regexp_replace( blb
                                , '^'
                                ,'['
                                ,1,0,'m'
                                ) blb
             from csv
         )
, jsn2 as (  -- Zeilenende
           select regexp_replace( blb
                                , '$'
                                ,']'
                                ,1,0,'m'
                                ) blb
             from jsn1
)
```

```
BLB
------------------------------------
["eins", "zwei", "drei"]
["vier", "fünf", "sechs"]
["sieben", "acht", "neun"]
```

# JSON Umwandlung: Kommas zwischen Zeilen

```
, jsn2 as (  -- Zeilenende
             select regexp_replace( blb
                                  , '$'
                                  ,']'
                                  ,1,0,'m'
                                  ) blb
               from jsn1
)
, jsn3 as (  -- Kommas zwischen Zeilen
             select regexp_replace('['||blb||']'
                                  , '\]'||chr(10)||'\['
                                  , '],['
                                  ,1,0,''
                                  ) blb
               from jsn2
)
```

```
BLB
--------------------------------------------------------------------------------
[["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]
]
```

# Das konvertierte JSON abfragen

```
with
  csv as (
           select to_clob( '"eins", "zwei", "drei"'||chr(10)
                         || '"vier", "fünf", "sechs"'||chr(10)
                         || '"sieben", "acht", "neun"'||chr(10)
                          ) blb
             from dual
         )
, jsn1 as ( -- Zeilenanfang
--...
, jsn2 as ( -- Zeilenende
-- ...
, jsn3 as ( -- Kommas zwischen Zeilen
-- ...
select zeile
     , spalte
     , wert
     , blb
  from jsn3
     , json_table( blb
                 , '$[*]'
                 columns ( zeile for ordinality
                         , NESTED PATH '$[*]'
                           COLUMNS ( wert varchar2 PATH '$'
                                   , spalte for ordinality
                                   )
                         )
                 )
order by zeile
;
```

# CSV einlesen: Zwischenergebnis

| ZEILE | SPALTE | WERT | BLB |
|---|---|---|---|
| 1 | 1 | eins | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 1 | 2 | zwei | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 1 | 3 | drei | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 2 | 1 | vier | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 2 | 2 | fünf | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 2 | 3 | sechs | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 3 | 1 | sieben | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 3 | 2 | acht | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |
| 3 | 3 | neun | [["eins", "zwei", "drei"],["vier", "fünf", "sechs"],["sieben", "acht", "neun"]] |

9 rows selected.

# Pivotieren - SQL-Style

```
select zeile
      , max(decode(spalte,1, wert)) spalte_1
      , max(decode(spalte,2, wert)) spalte_2
      , max(decode(spalte,3, wert)) spalte_3
from(
with
  csv as (
-- ...
)
group by zeile
;
```

# Pivotieren mit der Magie von JSON_TABLE

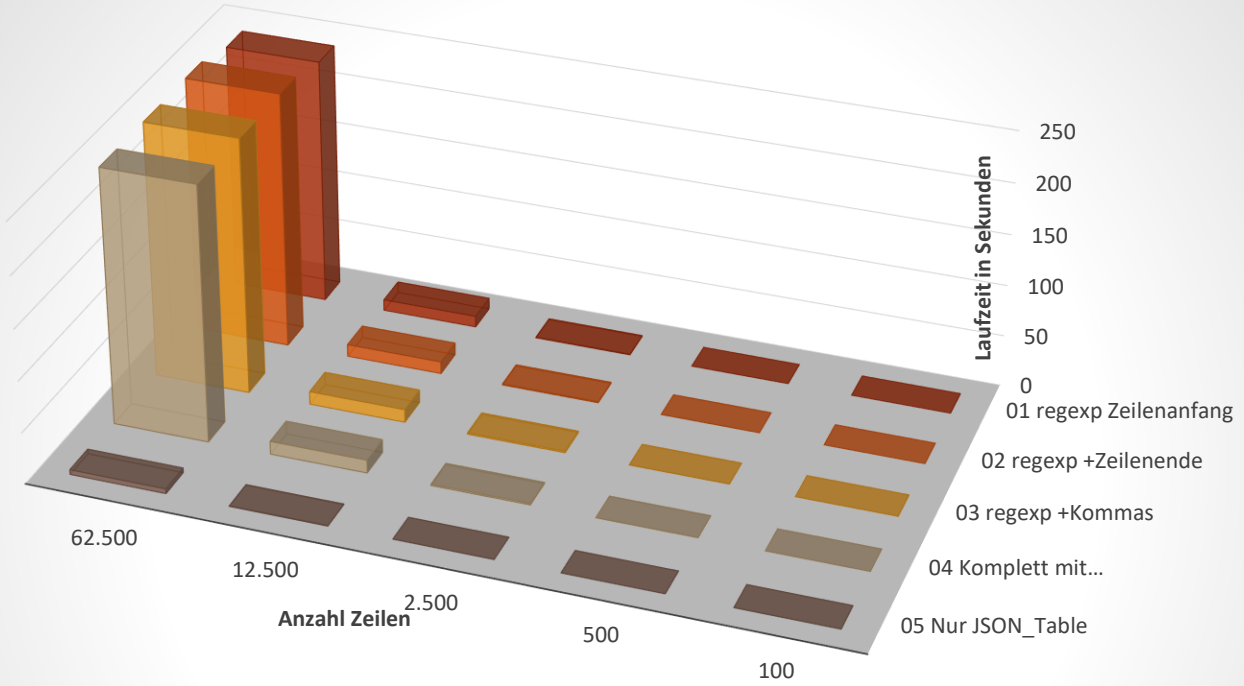
```
with
  csv as (
           select to_clob( '"eins", "zwei", "drei"'||chr(10)
                         || '"vier", "fünf", "sechs"'||chr(10)
                         || '"sieben", "acht", "neun"'||chr(10)
                         ) blb
             from dual
         )
, jsn1 as (  -- Zeilenanfang
-- ...
, jsn2 as (  -- Zeilenende
-- ...
, jsn3 as (  -- Kommas zwischen Zeilen
-- ...
select zeile, spalte1, spalte2, spalte3
  from jsn3
     , json_table( blb
                 , '$[*]'
                 columns ( spalte1 varchar2 PATH '$[0]'
                         , spalte2 varchar2 PATH '$[1]'
                         , spalte3 varchar2 PATH '$[2]'
                         , zeile for ordinality
                         )
                 )
;
```

# CSV einlesen: Endergebnis

```
32  )
33  select zeile, spalte1, spalte2, spalte3
34    from jsn3
35       , json_table( blb
36                   , '$[*]'
37                   columns ( spalte1 varchar2 PATH '$[0]'
38                           , spalte2 varchar2 PATH '$[1]'
39                           , spalte3 varchar2 PATH '$[2]'
40                           , zeile for ordinality
41                           )
42*                  )

    ZEILE SPALTE1         SPALTE2         SPALTE3
--------- --------------- --------------- ---------------
        1 eins            zwei            drei
        2 vier            fünf            sechs
        3 sieben          acht            neun
```

# Performancebetrachtung




| | 100 | 500 | 2.500 | 12.500 | 62.500 |
|---|---|---|---|---|---|
| 01 regexp Zeilenanfang | 0,04 | 0,14 | 0,84 | 10,98 | 239,96 |
| 02 regexp +Zeilenende | 0,03 | 0,18 | 1,11 | 11,91 | 249,76 |
| 03 regexp +Kommas | 0,03 | 0,20 | 1,30 | 12,62 | 249,15 |
| 04 Komplett mit JSON_Table | 0,04 | 0,20 | 1,33 | 12,70 | 248,93 |
| 05 Nur JSON_Table | 0,01 | 0,02 | 0,08 | 0,47 | 4,60 |

# Blobs vom Client in die Datenbank laden

SQLcl ist das neue SQL*Plus

Scripts in JavaScript möglich

- its-people Vortrag Mittwoch, 16.11. 14:00

Beispiel

- Laden von Blobs

```
var HashMap = Java.type("java.util.HashMap");
var bindmap = new HashMap();

// Wir erwarten ein Argument: Den Dateinamen
ctx.write("Lese Datei: "+ args[1] + "\n");
var filePath=args[1];

var blob=conn.createClob();
var bstream=blob.setAsciiStream(1);
/* den Blob einlesen */
java.nio.file.Files.copy( java.nio.file.FileSystems.getDefault().getPath(
    filePath)
                        , bstream );
bstream.flush();

bindmap.put("csv",  blob);
bindmap.put("pfad", filePath);
if(!util.execute( "insert into csv_tab(csv,pfad) values(:csv, :pfad)"
                , bindmap)
  ){ ctx.write("insert fehlgeschlagen exit.\n");
     exit;
}
sqlcl.setStmt( "commit; \n"
             + "set sqlformat ansiconsole \n"
             + " select pfad,dbms_lob.getlength(csv) "
             + "from csv_tab;");
sqlcl.run();
```

Demo\12a_Blob_einlesen_mit_sqlcl.js

# CSV einlesen - Demo

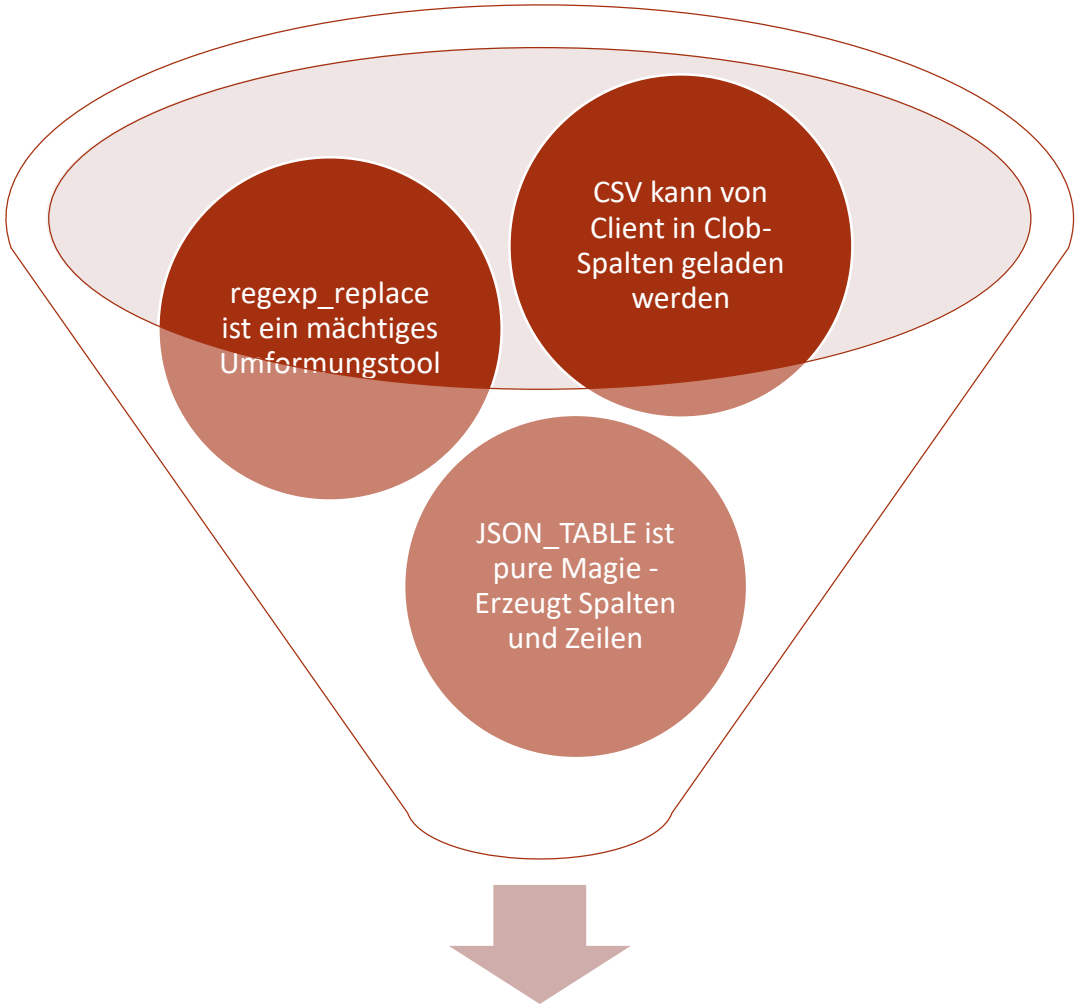
DEMO

# Ressourcen

## Quellen für Folien und Demos:

- DOAG Website / Konferenzplaner
- http://www.its-people.de/doag-2016
- https://github.com/its-people/csv-json_table

# Fazit




PL/SQL ist gut. SQL ist besser.

# Herzlichen Dank für Ihre Aufmerksamkeit !

we make the difference
www.its-people.de

# Fragen ?





**its-people GmbH**

| | |
|---|---|
| Frankfurt | Tel. 069 2475 2100 |
| Hamburg | Tel. 040 2360 8808 |
| Köln | Tel. 0221 1602 5204 |
| München | Tel. 089 5484 2401 |

**its-people ERP Beratungsgesellschaft mbH**

| | |
|---|---|
| Frankfurt | Tel. 069 2475 1980 |

**www.its-people.de** **info@its-people.de**