itspeople

Parsing CSV-Files with pure SQL & the magic of JSON_TABLE

9th February 2017

Robert Marz

Robert Marz

Client	Senior Technical Architect with database centric view of the world
its-people	Portfolio Manager Database Technologies Blog Editor
DOAG	Active Member Database Community in charge of Cloud topics

@RobbieDatabee

blog.its-people.de

Robert.Marz
@its-people.de

Motivation

CSV Files are everywhere

PL/SQL is cumbersome

Ask Tom Question:

parsing a CLOB field which contains CSV data

CSV File Format

Historically matured

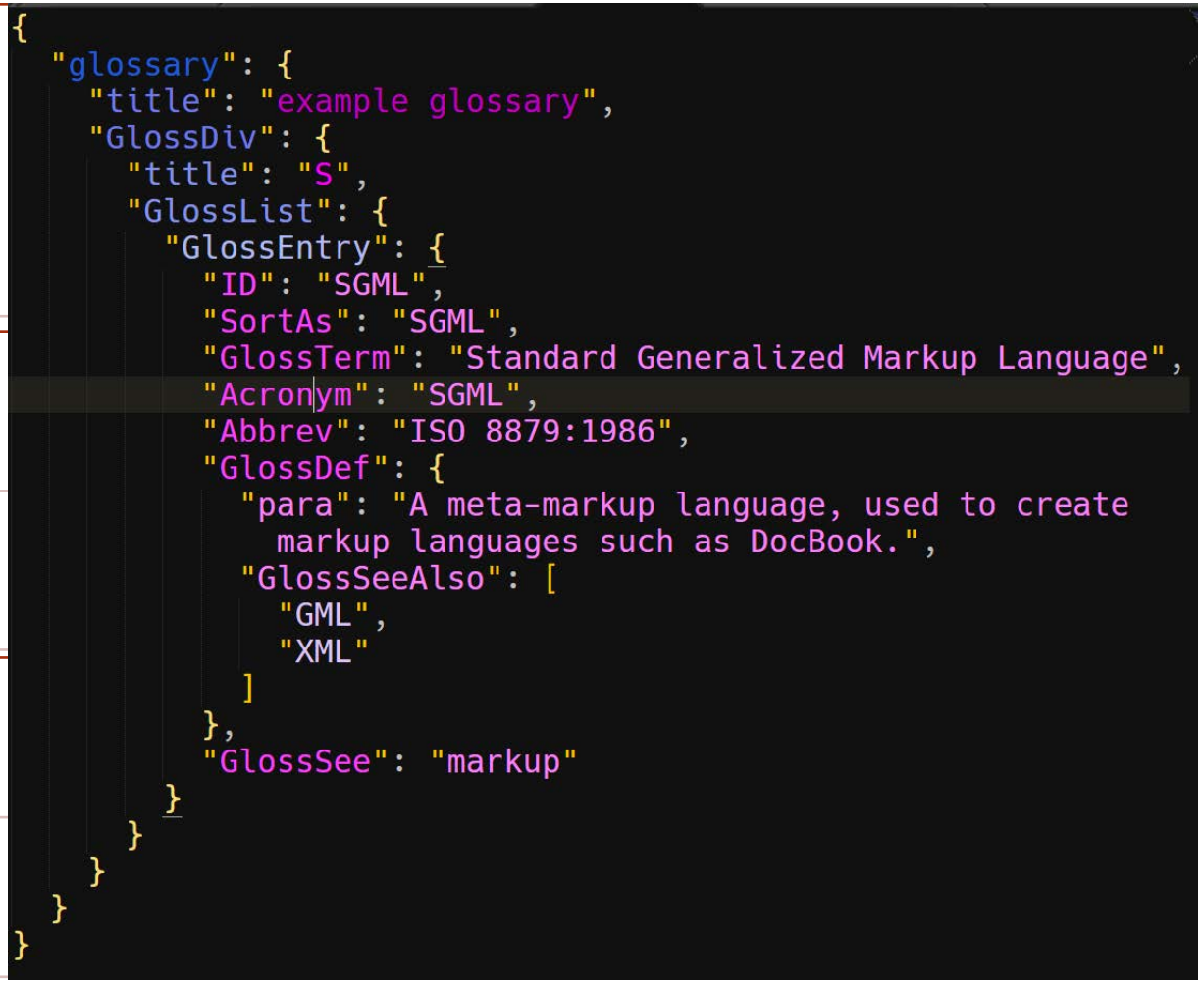
```
Eins,Zwei,Drei
Vier,Fünf
Sechs,Sieben,Acht
```

Standardized (a little):

- RFC4180 for CSV

JSON Files

Java **S**cript **O**bject **N**otation	“Key“:“Value“ Pairs
Think of XML with <Tags> replaced by brackets	{} – Groupings
	[] - Arrays
Schemaless	no constraints
	Developers hell



JSON Datatypes

JSON		
	String	"text" : "Hello world"
	Number	"integer" : 12345
		"double" : 123.45
		"float" : 1.234e-6
	Boolean	„Wahr" : true
		„Falsch" : false
		„Dunno" : null
	RFC7159 for JSON	

Comparing CSV to JSON

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

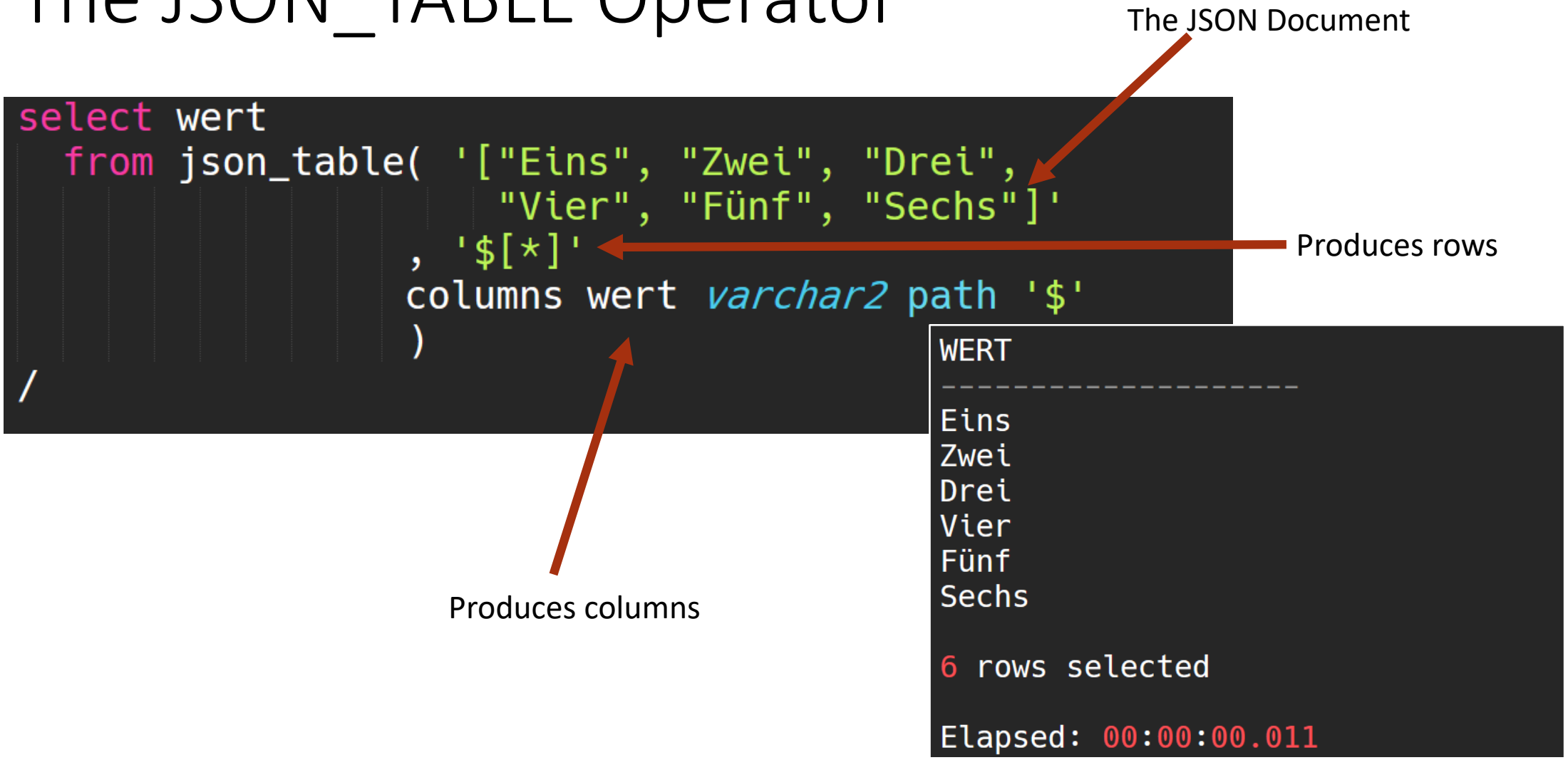
```
["Eins", "Zwei", "Drei"
,"Vier", "Fünf"
, "Sechs", "Sieben", "Acht"]
```

The JSON_TABLE Operator

JSON_TABLE

- Lives inside the SQL-From-Clause
- Produces **Rows** and **Columns**
- Accepts CLOBs with JSON data
- To be included in SQL standard

The JSON_TABLE Operator



Transposing CSV to JSON

Regular Expressions

Requirement

preserve rows and columns

Solution

nested JSON Arrays

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

Regular Expressions - brief overview

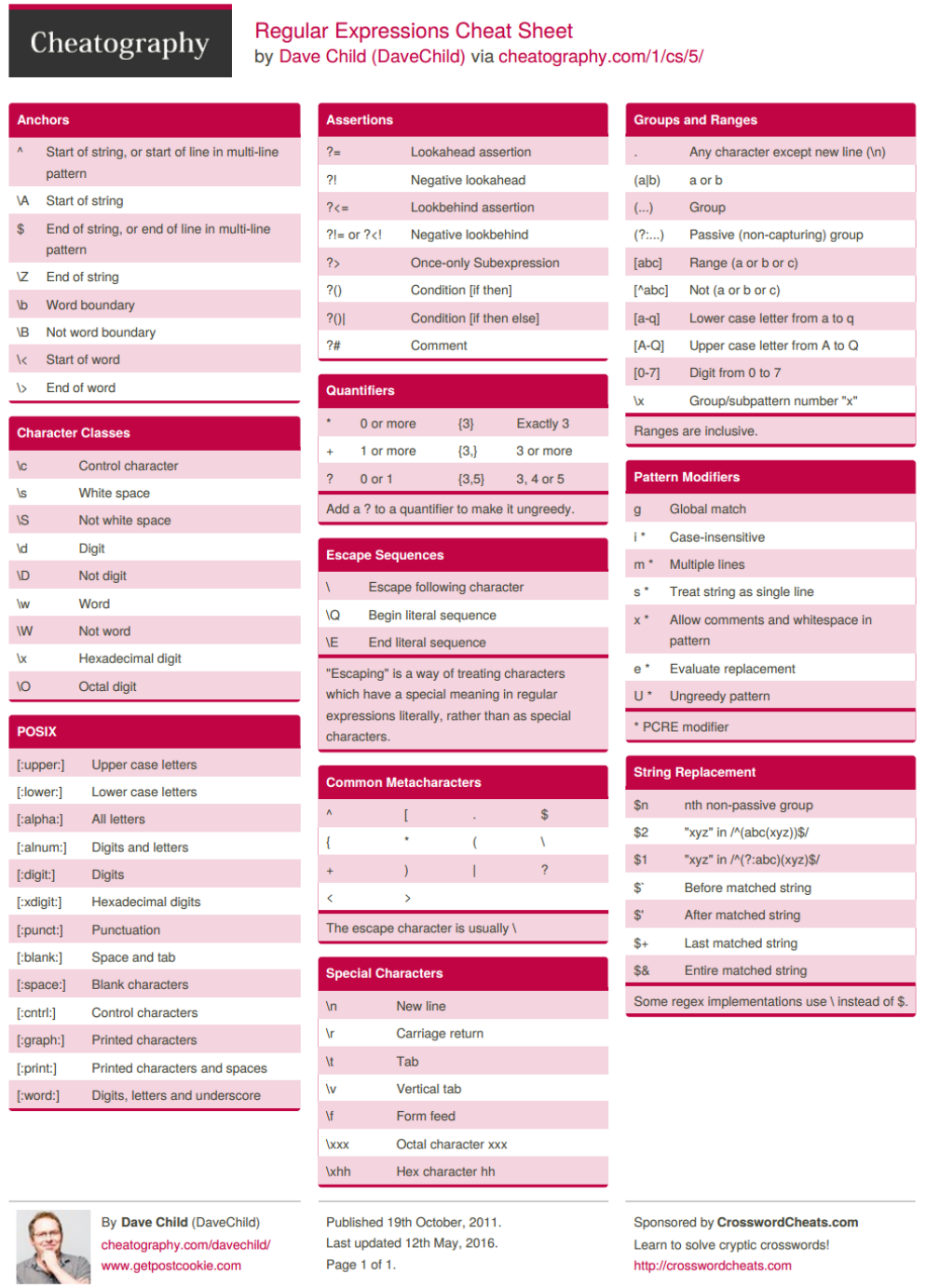
RegExp short and simple won´t work.

Google:
RegExp Tutorial|Introduction|Tips

Your Oracle Rx:
Regular Expressions in an Oracle World

- Talk by Rumpi Gravenstein
- 1:30pm today Room 4e

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

The REGEXP_REPLACE Function

REGEXP_REPLACE	Lives inside	SQL-Select-List
		Where-Clause
	Works like replace()	regular expressions instead of static strings
	Accepts	CLOBs

The REGEXP_REPLACE Function

```
select regexp_replace( 'Eins,Zwei,Drei' ||chr(10)
                     || 'Vier,Fünf'     ||chr(10)
                     || 'Sechs,Sieben,Acht'
                      , '^|$'
                      , '"'
                      ,1,0
                      ,'m') csv
  from dual
/
```

Source Document

Search RegExp Pattern

Replace String

Start Position & Number Occurences

Match Parameter:

„m“:	Treat as multiple lines
„i“, „c“:	Case (i)nsensitive
„n“: „.“	matches newline
„x“:	ignore whitespace

```
CSV
--------------------

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

JSON Transformation: Array – start of line

```
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 Transformation: Array – end of line

```
, 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 Transformation: Placing commas

```
, 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"]
]
```

Querying the transformed JSON

```
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
;
```

Parsing CSV: Intermediate Result

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.

Pivoting – SQL Style

```
    ZEILE SPALTE_1              SPALTE_2              SPALTE_3
--------- --------------------- --------------------- --------------------
        1 eins
        1                       zwei
        1                                             drei
        2 vier
        2                       fünf
        2                                             sechs
        3 sieben
        3                       acht
        3                                             neun

9 Zeilen gewählt.
```

Pivoting – the magic of 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
                         )
                 )
;
```

Parsing CSV: Final Outcome

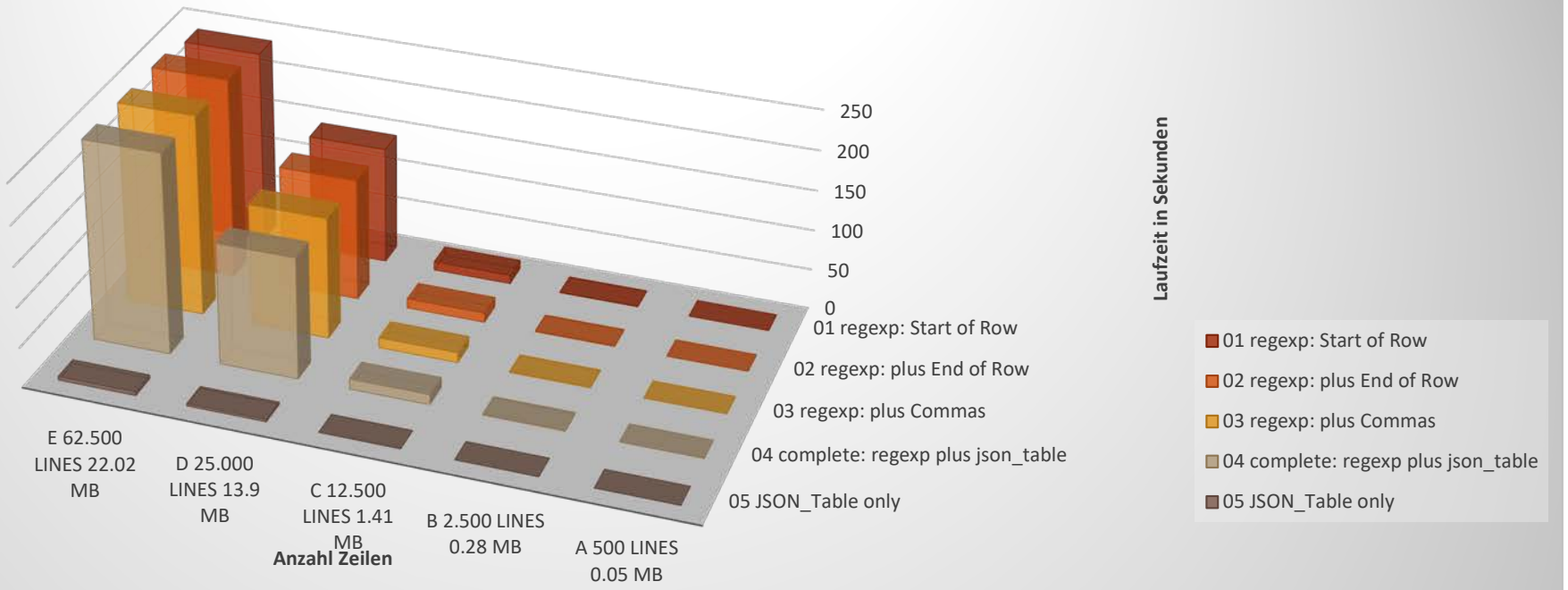
```
32  )
33  select zeile, spalte1, spalte2, spalte3
34    from jsn3
35       , json_table( blb
36                   , '$[*]'
37                   columns ( spalte1 varchar2 PATH '$[0]'
38                           , spalte2 varchar2 PATH '$[1]'
39                           , spalte3 varchar2 PATH '$[2]'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

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

Performance Evaluation

Oracle 12.1 running inside a VirtualBox VM on a Tablet-PC.

Your Milage may vary



	a 500 lines 0.05 MB	b 2.500 lines 0.28 MB	c 12.500 lines 1.41 MB	d 25.000 lines 13.9 MB	e 62.500 lines 22.02 MB
01 regexp: Start of Row	0,14	0,84	10,98	145,43	239,96
02 regexp: plus End of Row	0,18	1,11	11,91	151,37	249,76
03 regexp: plus Commas	0,20	1,30	12,62	151,00	249,15
04 complete: regexp plus json_table	0,20	1,33	12,70	150,87	248,93
05 JSON_Table only	0,02	0,08	0,47	2,79	4,60

Loading Blobs from Client into the Database

SQLcl ist the new SQL*Plus

Scripting with JavaScript

- See Slides from my yesterday Talk

Example:

- Loading 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

Loading CSV - Demo

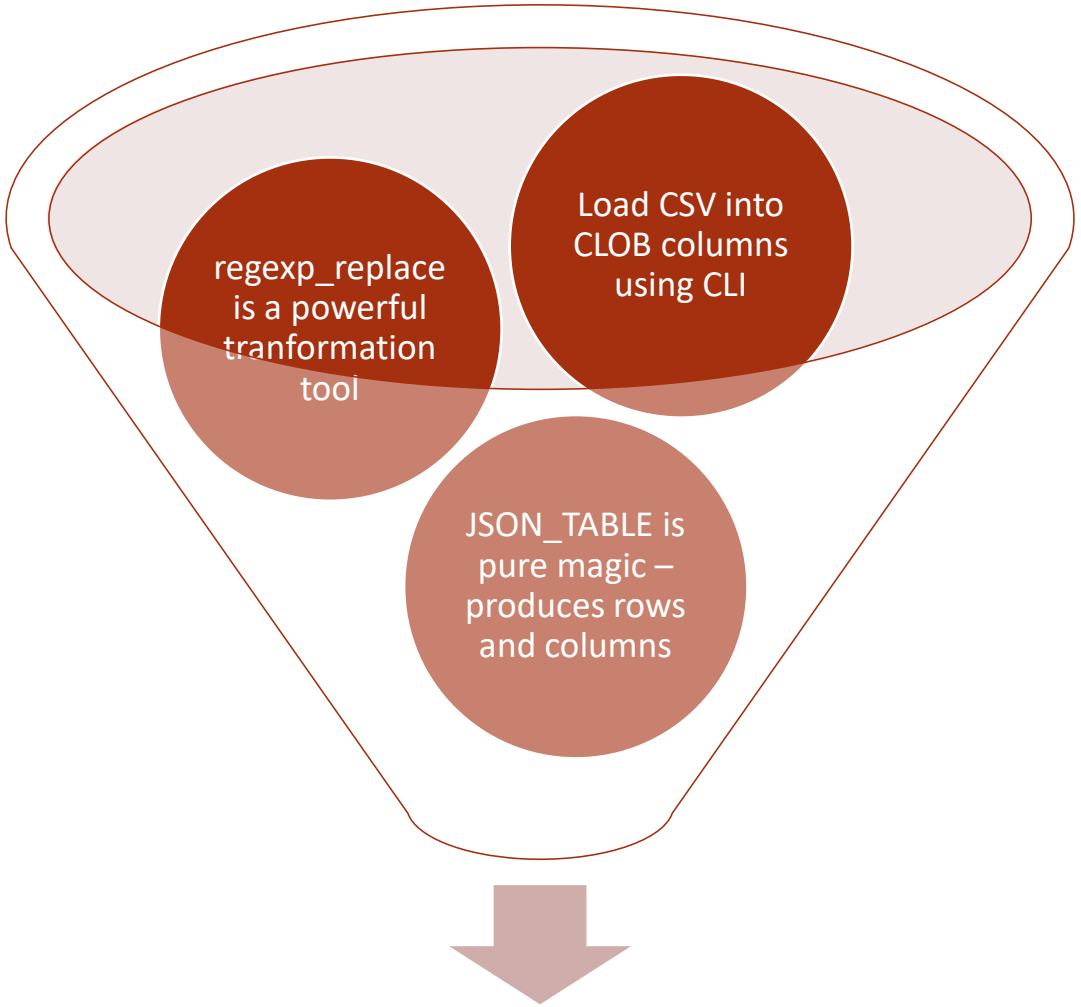
DEMO

Ressources

Download Slides and Demos

- RMOUG Website
- GitHub: https://github.com/its-people/csv-json_table

Bottom Line



PL/SQL is good. Pure SQL is better.

Herzlichen Dank für Ihre Aufmerksamkeit!

we make the difference
www.its-people.de

Questions?

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**