Your Database Can Do Complex String Manipulation Too!

Harry Droogendyk – Stratia Consulting Inc.

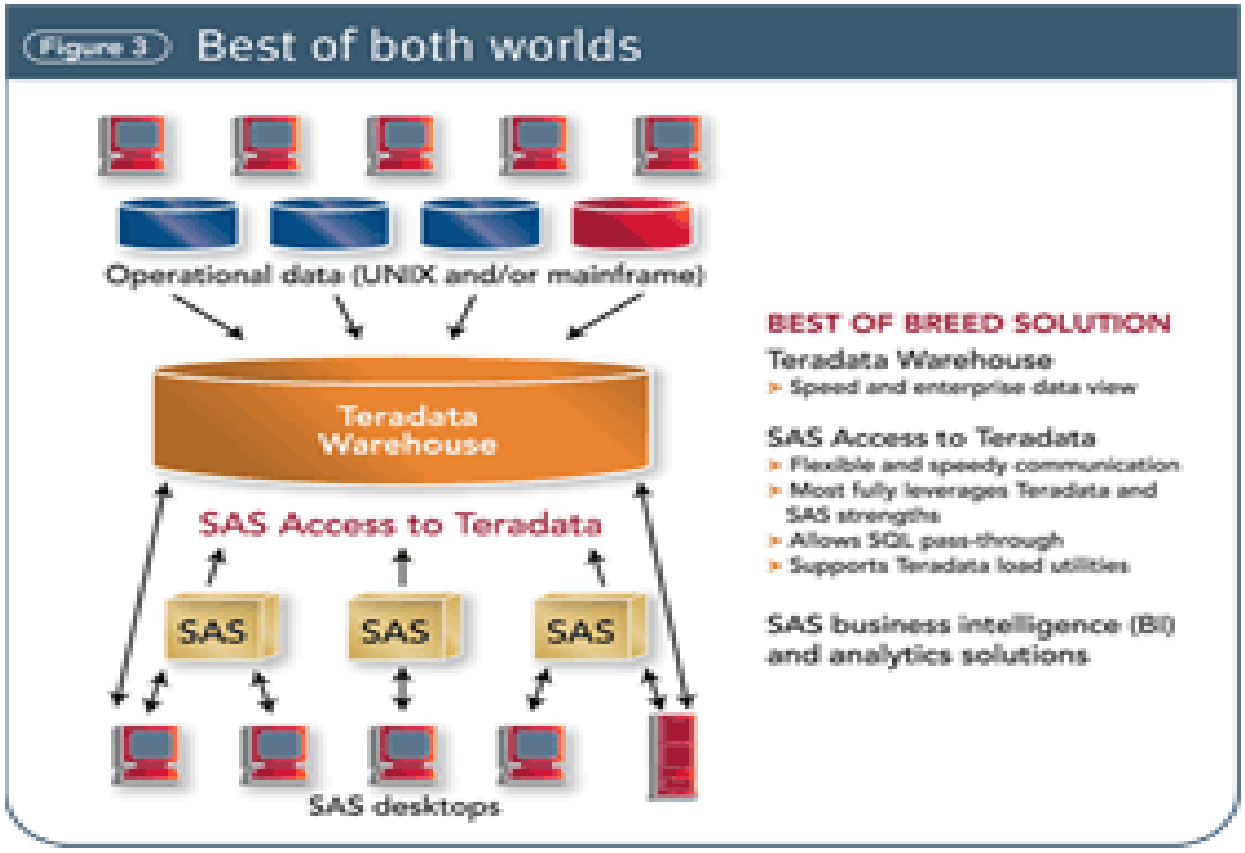
Data Processing

- numeric data is easy
  - standard functions
- character data increasingly valuable
  - unstructured
    - Big Data
  - text mining
  - looking for valuable nuggets !
  - not so easy
- requires parsing

Character Functions

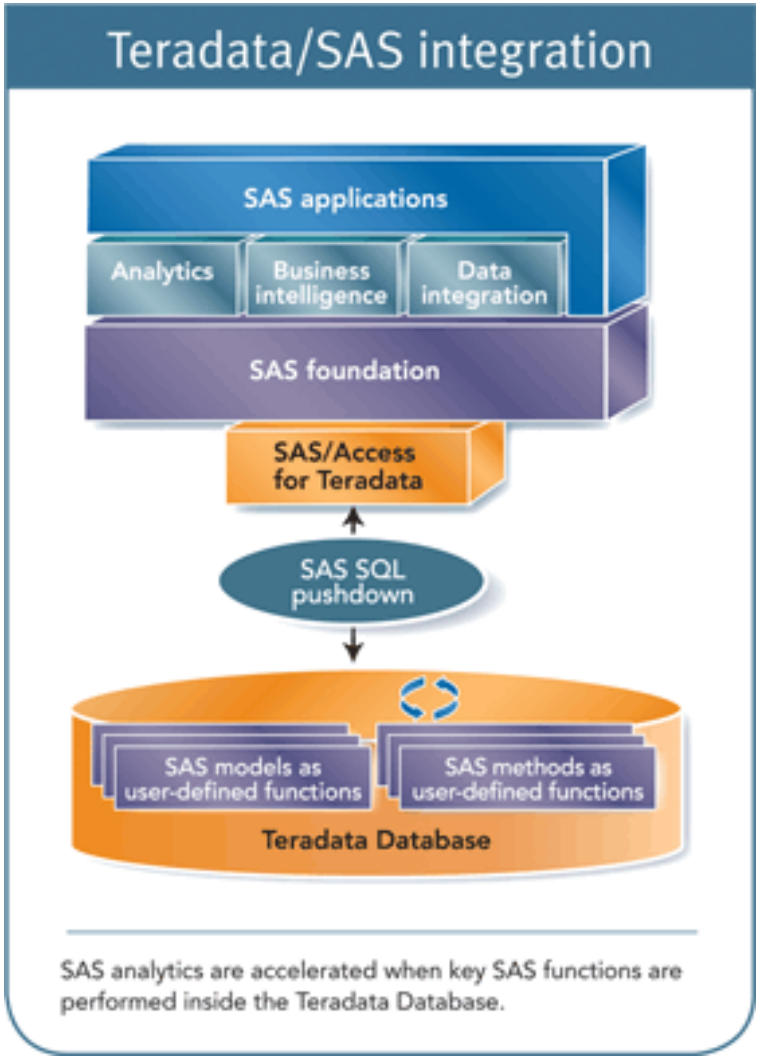
- SAS has 96 character functions in v9.4
- functions available in SQL
  - Teradata v15 30
  - Oracle 12c 40
- SAS data step language
  - conditional processing
  - looping
- how to process complex character data?

Typical Architecture

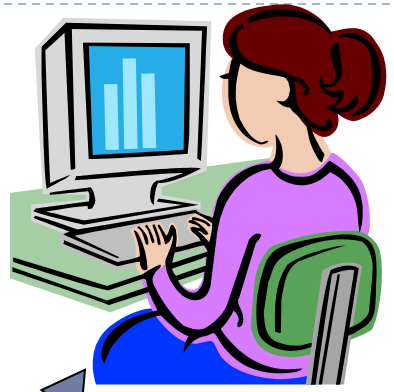


In-Database Processing

- SAS Analytics pushed down into the database
- implicit pass-thru
  - SAS Access libname
  - PROCs like FREQ create SQL queries
- DS2
- Not everything translates to SQL



Typical Architecture



why is the SAS server dying?!

Regular Expression Functions

- SAS and regex
  - RX*
  - PRX*
- SQL regex is relatively recent vintage
  - Oracle 10g
  - Teradata v14
- read the "what's new" section in docs?

Regular Expression Functions

- Robert Frost

  I shall be telling this with a sigh
  Somewhere ages and ages hence:
  Two roads diverged in a wood, and I—
  I took the one less traveled by,
  And that has made all the difference.

- regex is the road less traveled...

KISS

- SQL *do*es have character functions
  - many identical or similar in function
  - many named the same
  - some not found in SAS
  - keep it simple if you can

- Teradata examples:
  - TRIM INDEX SUBSTR LENGTH
  - (NOT) LIKE
  - TRANSLATE → oTRANSLATE
  - TRANWRD → oREPLACE
  - SCAN → STRTOK

Enhanced Functionality Required

- checking if a string is numeric
  - Netezza isNumeric
  - Teradata TO_NUMBER
- pattern matching
  - alpha numeric
  - phone number
  - email address
- pattern anywhere in the string
- character replacement
  - characters (not) adjacent to specified pattern
- extracting unknown number of values

Regular Expressions

- special text for describing a search pattern
  - wildcard on steroids
- tokens
  - describes character types
- anchors
  - location
- qualifiers
  - quantity

- different dialects of regex

Regex Tokens

Token	Meaning
.	any character
\s	any whitespace
\d, [:digit:]	any digit
[abc]	single character: a, b or c
[^abc]	character other than a, b or c
[a-zA-Z]	all characters in the range specified
\w	any word separated by whitespace
(dog|cat)	either "dog" or "cat"
(...)	capture everything enclosed
\	escape character, use when searching for tokens, qualifiers, ? + * $ ^

Regex Anchors

Anchor	Meaning
^	start of string
$	end of string
\b	word boundary (not supported by Oracle)

Regex Qualifiers

Qualifier	Meaning
?	zero or one
*	zero or more
+	one or more
{3}	exactly 3 of whatever precedes, e.g. \d{3}
{3,6}	between 3 and 6 of whatever precedes, e.g. [a-z]{3,6}

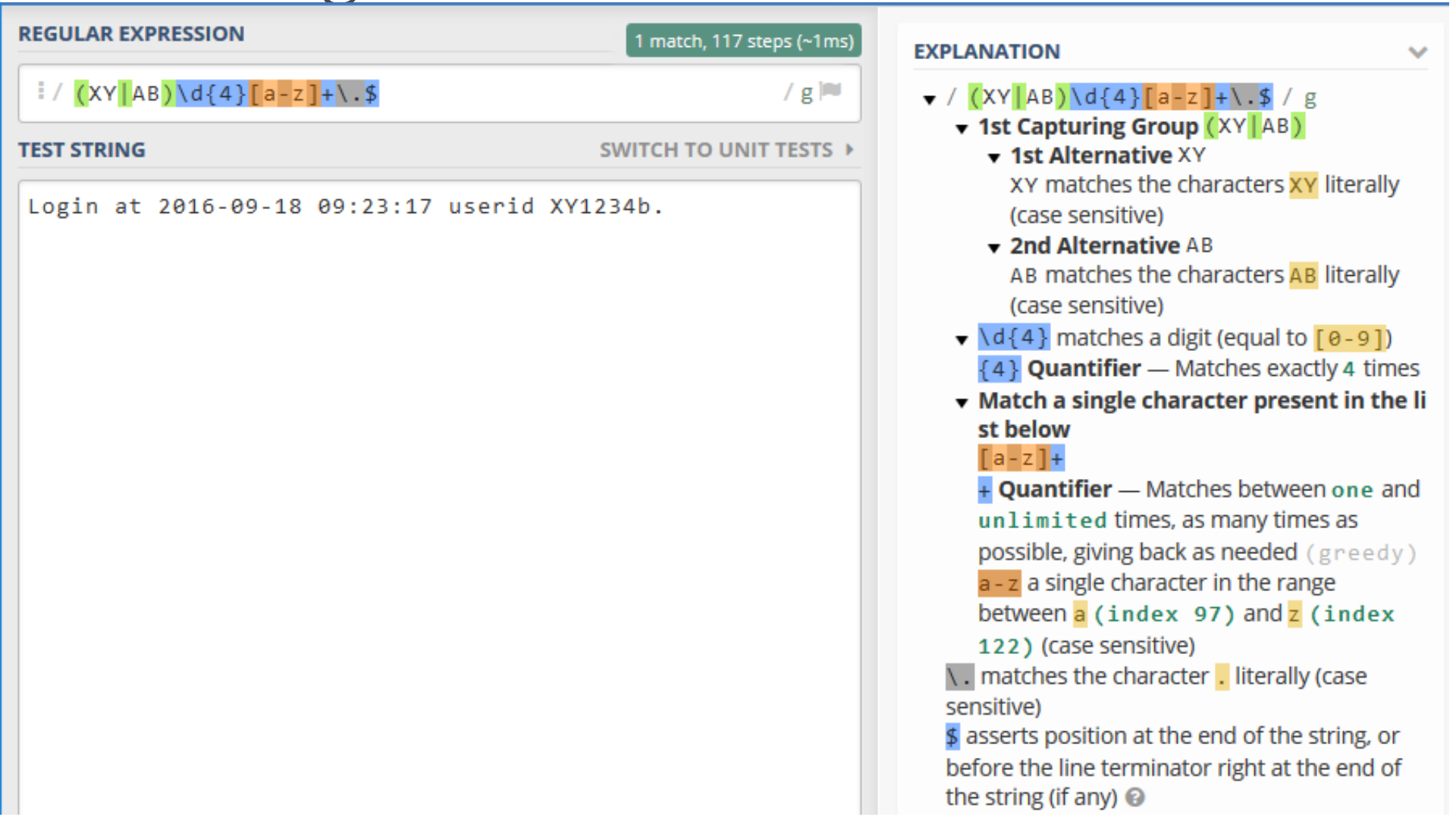
Regex Example

(XY|AB)\d{4}[a-z]+\.$

Section	Meaning
(XY|AB)	literal XY or AB
\d{4}	exactly four digits
[a-z]+	one or more lower case alphabetic characters
\.	literal period, escaped because period is also a token meaning any character
$	at the end of the string

`Login at 2016-09-18 09:23:17 userid` **`XY1234b.`** **`- match`**
`AB1234x.` `is the user's ID` - nomatch

www.regex101.com



Caveats

- ANSI SQL is a myth
  - consult DB docs
- different DBs support different regex dialects
  - Oracle POSIX
  - Teradata Perl
- CAST results to set column width
  - Teradata defaults to varchar(8000)
- examples tested in Teradata
  - YMMV

Regex Functions

Function	Use
REGEXP_INSTR	find starting or ending position of a string pattern in the source
REGEXP_SIMILAR / LIKE	Boolean result - is string pattern in source?
REGEXP_SUBSTR	extract a portion of source that matches string pattern
REGEXP_REPLACE	replace a portion of source that matches string pattern
REGEXP_SPLIT_TO_TABLE	split delimited string into rows, delimiter defined by regex pattern

REGEXP_INSTR

- find starting or ending position of string
  - extends vanilla INSTR

```
REGEXP_INSTR    ( source, regex, start pos,
  occur, return, match )
```

Parameter	Use / Values
source	column or literal to be searched
regex	regex pattern
start position	position in source to begin searching, relative to 1, default is 1
occurrence	occurrence of string matching regex pattern, default is 1
return	0=starting position of matched string, 1=position following end of matched string, default is 0
match	i=ignore case, c=case sensitive (additional, less commonly used values allowed as well)

REGEXP_INSTR Example 1

- find position of zip code

```
SELECT REGEXP_INSTR   (
  'Joe Smith, 10045 Berry Lane, San
  Joseph, CA 91234',
    '[[:digit:]]{5}$')
```

Section	Meaning
`[[:digit:]]`	digits, 0-9 characters
`{5}`	exactly 5 of preceding
`$`	anchor at end, grab last 5

REGEXP_INSTR Example 2

- find position *following* non-alphabetic characters

```
SELECT REGEXP_INSTR(
'123ABC','[^a-z]{3}',1,1,1,'i') -- 4
```

Section	Meaning
`[^a-z]{3}`	exactly 3 non-alphabetic
`1,1,1`	starting position, 1st occurence, return ending position
`'i'`	case insensitive

REGEXP_SIMILAR / LIKE

- returns Boolean result if pattern is found
  - Teradata requires the *entire* string match the regex pattern

```
REGEXP_SIMILAR ( source, regex, match )
```

Parameter	Use / Values
source	column or literal to be searched
regex	regex pattern
match	i=ignore case, c=case sensitive (additional, less commonly used values allowed as well)

REGEXP_SIMILAR

- WHERE clauses
- CASE statements

```
SELECT * FROM schema.table
 WHERE REGEXP_SIMILAR(char_column,
  'regex', 'i' ) = 1

SELECT CASE WHEN
   REGEXP_SIMILAR
      (char_column, 'regex', 'i' )
           THEN 'Y' END AS CONTACT_FLAG
   FROM schema.table
```

REGEXP_SIMILAR Example 1a

- programmer userid in the string
  - find a word made up of "pg" followed by 5 digits – returns 0 ?!?

```
SELECT REGEXP_SIMILAR
       ('Userid is pg26581',
       '\bpg\d{5}\b') -- 0
```

Section	Meaning
\b	word boundary
pg	literal 'pg'
\d{5}	five digits
\b	word boundary

REGEXP_SIMILAR Example 1b

- programmer userid in the string
  - find a word made up of "pg" followed by 5 digits – works now!

```
SELECT REGEXP_SIMILAR
       ('Userid is pg26581',
       '.*\bpg\d{5}\b.* ') -- 1
```

Section	Meaning
.*	zero or more of any character
\b	word boundary
pg	literal 'pg'
\d{5}	five digits
\b	word boundary
.*	zero or more of any character

REGEXP_SIMILAR Example 1c

- programmer userid in the string
  - find a word made up of "pg" followed by 5 digits – returns 1 ?!?

```
SELECT REGEXP_SIMILAR
       ('Userid is apg26581a',
       '.*pg\d{5}.*') -- 1
```

Section	Meaning
.*	0+ of any character
pg	literal 'pg'
\d{5}	five digits
.*	0+ of any character

REGEXP_SIMILAR Example 1 - Oracle

- programmer userid in the string
  - find a word made up of "pg" followed by 5 digits

```
SELECT REGEXP_LIKE
       ('Userid is pg26581',
       '(\W|^|$)pg\d{5} (\W|^|$) ') -- 1
```

Section	Meaning
`(\W|^|$)`	beginning/end of string, or not a word character (alphanumeric)
`pg`	literal 'pg'
`\d{5}`	five digits
`(\W|^|$)`	beginning/end of string, or not a word character (alphanumeric)

REGEXP_SIMILAR Example 2

- select only rows containing a phone number

```
SELECT wo_no, notes_txt
    FROM schema.table
   WHERE REGEXP_SIMILAR ( notes_txt,
  '.*\d{3}[\s.-]?\d{3}[\s.-]?\d{4}.*')
```

REGEXP_SIMILAR Example 2

```
'.*\d{3}[\s.-]?\d{3}[\s.-]?\d{4}.*'
```

Section	Meaning
`.*`	0+ of any character
`\d{3}`	three digits
`[\s.-]?`	\s - whitespace .- - period or dash ? - 0 or 1 of stuff between [ ]

REGEXP_SIMILAR Example 2

WO_NO	NOTES_TXT	REGEX_FLG
323273660	16 06 25	0
610511207	ADD ULTRA HIGH SPEED ON 509-233-2307	1
904716413	*PLS ADD 7 MB BUS HS TO 901 852 8001	1
832417421	CONTACT #: 9028113934	1
392539413	ACTION REQ'D:CONFIRM GWI REFLECTS SOFFS/PARMS ADDED	0
818123613	TECHNOLOGY PORTING:FOLLOW-UP:1-NPACAC	0
858964600	MOVE ORDER ON 905-512-7228	1
312402251	16 06 22	0

REGEXP_SUBSTR

- extract the matched string
  - extends vanilla SUBSTR

```
REGEXP_SUBSTR ( source, regex, start
  pos, occur, match )
```

Parameter	Use / Values
source	column or literal to be searched
regex	regex pattern
start position	position in source to begin searching, relative to 1, default is 1
occurrence	occurrence of string matching regex pattern, default is 1
match	i=ignore case, c=case sensitive (additional, less commonly used values allowed as well)

REGEXP_SUBSTR Example 1

```
SELECT wo_no, notes_txt,

REGEXP_SUBSTR ( notes_txt,
  '\d{3}[\s.-]?\d{3}[\s.-]?\d{4}',1,1)
  AS phone1,

REGEXP_SUBSTR ( notes_txt,
  '\d{3}[\s.-]?\d{3}[\s.-]?\d{4}',1,2)
  AS phone2

   FROM schema.table
```

REGEXP_SUBSTR Example 2

```
attr_key =
  'Device.Hosts.Host.10.X_OUI_History.Layer1
  Interface'
REGEXP_SUBSTR (attr_key,
  '(?<=Host\.)[0-9]+') as host_no  -- 10
```

Section	Meaning
`(?<=Host\.)`	positive lookbehind – does "Host." precede the digits ?
`[0-9]+`	one or more digits

- find the digits
  - before accepting, are they preceded by 'Host.' ?
  - returns `10`

REGEXP_SUBSTR Example 3

```
attr_key =
  'Device.Hosts.Host.10.X_OUI_History.Layer1
  Interface'
REGEXP_SUBSTR(attr_key, '[^.]*$')
      as attr_key_last
```

Section	Meaning
`[^.]`	match any character other than period
`*`	zero or more times
`$`	assert position at end of string

- start at the *end* of the string
  - gobble zero or more characters until a period is encountered
  - returns `Layer1Interface`

REGEXP_REPLACE

- replace the matched string
  - with another string, pattern or null

```
REGEXP_REPLACE( source, regex, replacement,
start pos, occur, match )
```

Parameter	Use / Values
source	column or literal to be searched
regex	regex pattern
replacement	replacement string *or pattern*
start position	position in source to begin searching, relative to 1, default is 1
occurrence	occurrence of string matching regex pattern, **0=all**
match	i=ignore case, c=case sensitive (additional, less commonly used values allowed as well)

REGEXP_REPLACE Example 1

```
SELECT REGEXP_REPLACE
('519.647-2472','[^0-9]','',1,0)
      AS PHONE_NO;
```

Section	Meaning
[^0-9]	match any non-numeric
''	nuttin'
1	starting position
0	replace all occurences

- all phone number delimiters will be removed
  - all non-numeric
  - 5196472472 returned

REGEXP_REPLACE Example 2

```
New Brunswick/KEDGWICK/506283, 506284,
New Brunswick/BLACKVILLE/506586, 506843,
```

- commas between NPANXX values *and* between Province/City groups
- replace NPANXX comma separators with semi-colons so commas can be used to separate Province/City groups

```
New Brunswick/KEDGWICK/506283; 506284,
New Brunswick/BLACKVILLE/506586; 506843,
```

REGEXP_REPLACE Example 2

```
with test_data as (
     SELECT 'New Brunswick/KEDGWICK/506283, 506284,
             New Brunswick/BLACKVILLE/506586, 506843,'
                      as log_value )
SELECT REGEXP_REPLACE(log_value,'\,\s+(\d)',';\1',1,0)
  as fixed_value
   from test_data;
```

Section	Meaning
\,	match comma
\s+	one+ white space characters
(\d)	match a digit, (capture group)
;\1	replacement string, literal ; back reference to digit in capture group
1	starting position
0	replace all occurrences

REGEXP_REPLACE Example 2

```
New Brunswick/KEDGWICK/506283; 506284,
New Brunswick/BLACKVILLE/506586; 506843,

select    row_id
     ,    day_of_calendar            as prov_city_group_no
     ,    regexp_substr(fixed_value,'[^,]+',1, day_of_calendar)
                                     as prov_city_group_value
  from     fixed_separators  a,
          ( select day_of_calendar
             from sys_calendar.calendar
            where day_of_calendar between 1 and
                  ( select max(length(fixed_value) -
                               length(oreplace(fixed_value,',', NULL)))
                     from fixed_separators )
          )  b     -- sub-query to avoid large product join with CALENDAR

 where     b.day_of_calendar between 1 and
          (length(a.fixed_value) - length(oreplace(a.fixed_value, ',', NULL)))
```

	row_id	prov_city_group_no	prov_city_group_value
1	1234	1	New Brunswick/KEDGWICK/506283;506284
2	1234	2	New Brunswick/BLACKVILLE/506586;506843

REGEXP_SPLIT_TO_TABLE

- split a delimited string into chunks
  - akin to STRTOK_SPLIT_TO_TABLE but with regex pattern

	Primary Key	Count	Value
Original	1		123;456;789
After SPLIT_TO _TABLE	1	1	123
	1	2	456
	1	3	789

- output is limited to key value, count, extracted value
  - must always join back to original table

REGEXP_SPLIT_TO_TABLE

```
REGEXP_SPLIT_TO_TABLE
( source table outkey, source, regex, match )
     RETURNS
( outkey, parsed count, parsed value )
```

Parameter	Use / Values
source table outkey	source column to be used as output key
source	data to be parsed
regex	regex pattern defining delimiters
match	i=ignore case, c=case sensitive (additional, less commonly used values allowed as well)
outkey	output key column definition
parsed count	parsed count column definition
parsed value	parsed value column definition

REGEXP_SPLIT_TO_TABLE Example

```
New Brunswick/KEDGWICK/506283, 506284,
New Brunswick/BLACKVILLE/506586, 506843,
```

- commas between NPANXX values *and* between Province/City groups
- ~~replace NPANXX comma separators with semi-colons so commas can be used to separate Province/City groups~~

```
New Brunswick/KEDGWICK/506283; 506284,
New Brunswick/BLACKVILLE/506586; 506843,
```

REGEXP_SPLIT_TO_TABLE Example

```
with test_data as (
    SELECT 'abc' as pk, 1 as hee, 'def' as haw,
        'New Brunswick/KEDGWICK/506283, 506284,
         New Brunswick/BLACKVILLE/506586, 506843,' as log_value    )
select *
  from  table (
    REGEXP_SPLIT_TO_TABLE( test_data.pk
                         ,  test_data.log_value
                         , '\,\s+(?=[^0-9])', 'i')
   RETURNS ( pk                    varchar(6)   character set unicode
           ,  prov_city_group_no    integer
           ,  prov_city_group_value varchar(128) character set unicode
             )
                ) AS sg
```

REGEXP_SPLIT_TO_TABLE Example

```
REGEXP_SPLIT_TO_TABLE( test_data.pk
                     ,  test_data.log_value
                     , '\,\s+(?=[^0-9])', 'i')
   RETURNS ( pk                    varchar(6)   character set unicode
           ,  prov_city_group_no   integer
           ,  prov_city_group_value varchar(128) character set unicode
             )
                ) AS sg
```

Section	Meaning
test_data.pk	source column supplying output key
test_data.log_value	source column to be parsed
\,	match comma
\s+	one+ whitespace characters
(?=[^0-9])	positive lookahead, does non-digit character follow whitespace and comma?

REGEXP_SPLIT_TO_TABLE Example

```
REGEXP_SPLIT_TO_TABLE( test_data.pk
                     ,  test_data.log_value
                     , '\,\s+(?=[^0-9])', 'i')
   RETURNS ( pk                    varchar(6)   character set unicode
           ,  prov_city_group_no   integer
           ,  prov_city_group_value varchar(128) character set unicode
             )
                ) AS sg
```

Section	Meaning
pk	output key column definition
prov_city_group_no	parsed count column definition
prov_city_group_value	parsed value column definition

Conclusion

- paper appendix has complete example

- SAS world is comfortable
- streeeettcccchhhhhh our minds
  - efficiency
  - right tool in the right place

- regex
  - write once, read never → nooooooooooo
  - document *each* token, anchor and qualifier

Conclusion

- own your ignorance ☺
  - www.regular-expressions.info
  - www.regexbuddy.com
  - www.regex101.com
    - poke and hope heaven !!

Dieter Noeth

0 Questions

This user has not asked any questions

Author

Harry Droogendyk
Stratia Consulting Inc.

www.stratia.ca/papers