Being Regular with Regular Expressions

John Garmany

Session

BURLESON CONSULTING

Who Am I

ORACLE | CERTIFIED PROFESSIONAL

John Garmany
Senior Consultant
Burleson Consulting

- West Point Graduate – GO ARMY!
- Masters Degree Information Systems
- Graduate Certificate in Software Engineering

Definition of Regular Expressions

Formally defined by information theory as defining the languages accepted by finite automata Source Neal Ford.

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions. Source grep man page

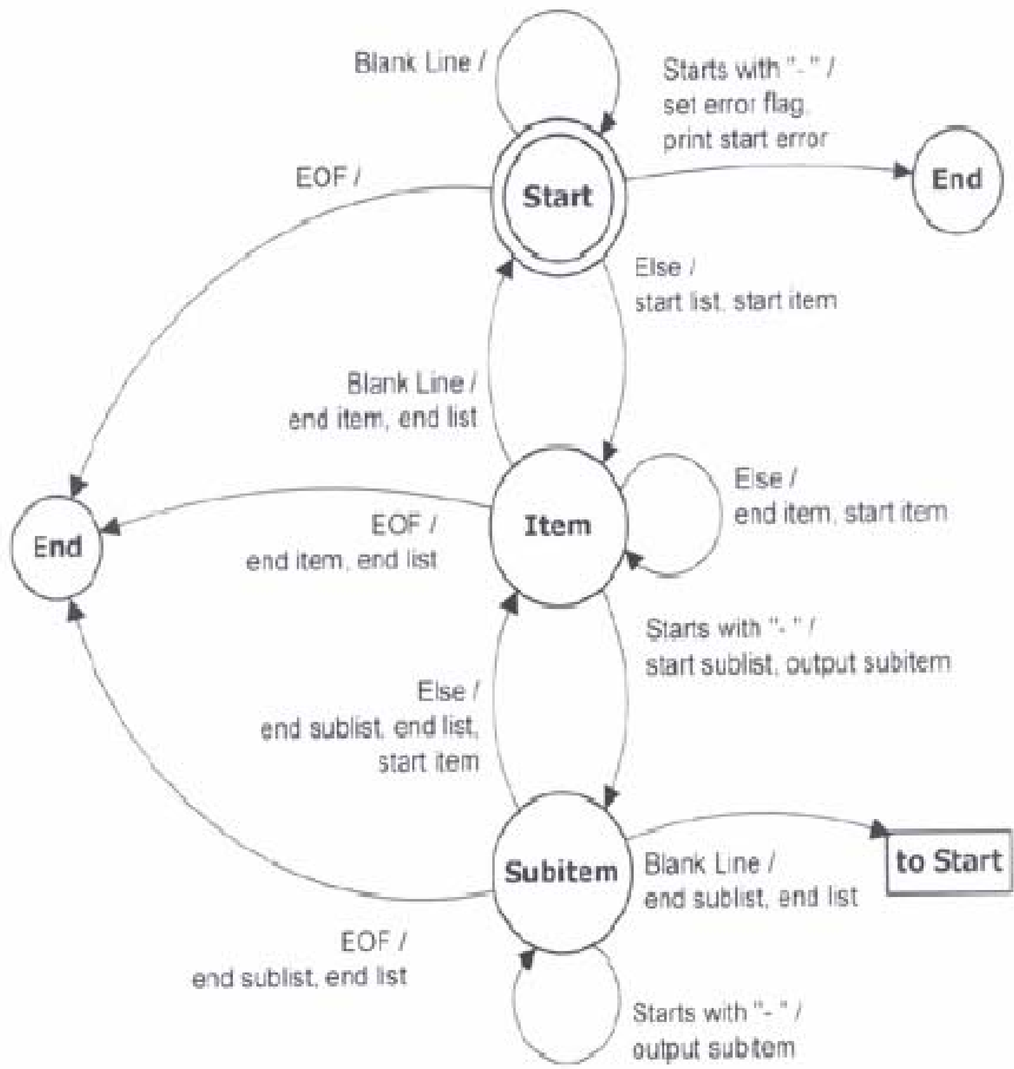
BURLESON
CONSULTING

Regular Expressions

Used for String Pattern Matching

Actually is a Formal State Machine.

The match is true if you finish in an acceptable state.



BURLESON
CONSULTING

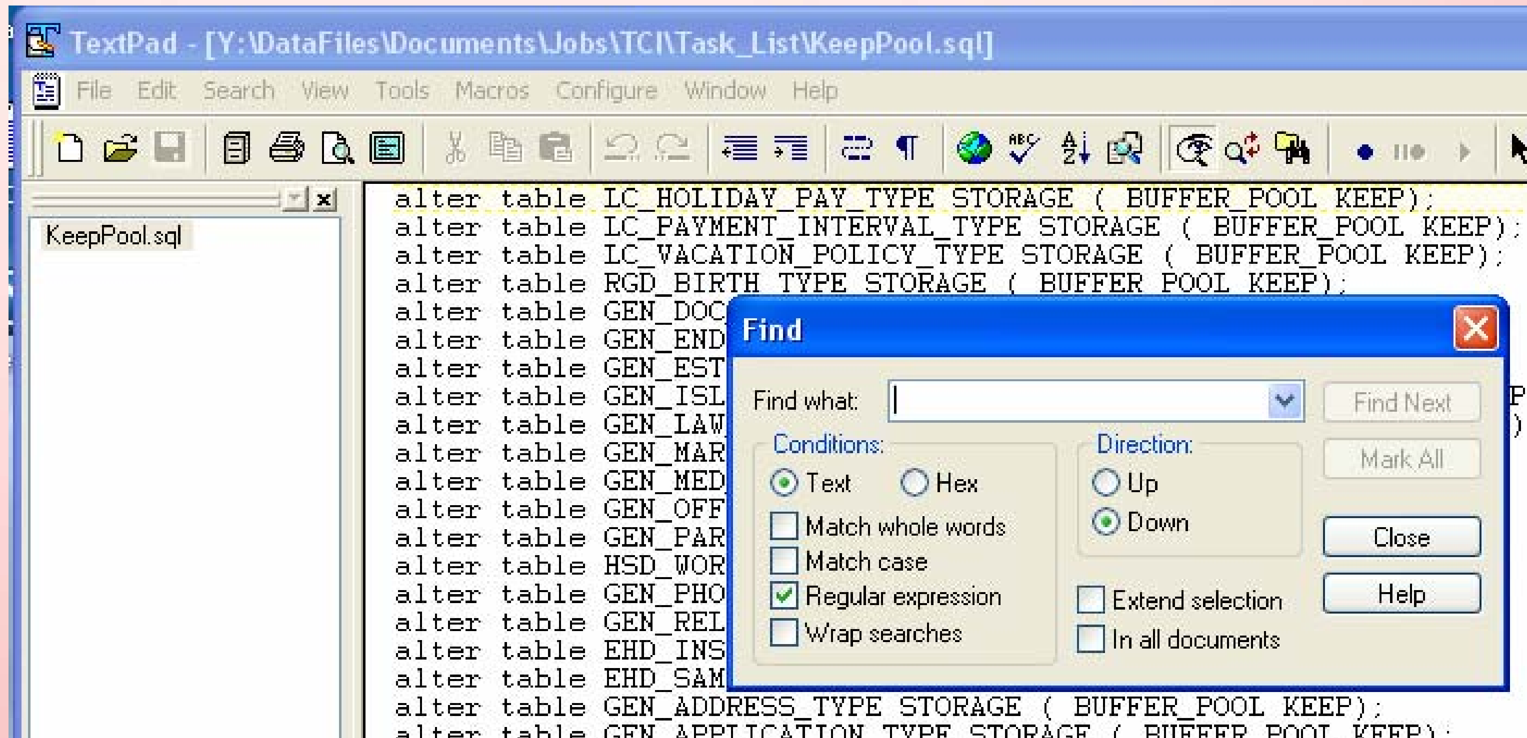
Unix and Regular Expressions

Many Unix utilities use Regular Expressions

- Grep = Global Regular Expression Print
- Sed = Stream Editor

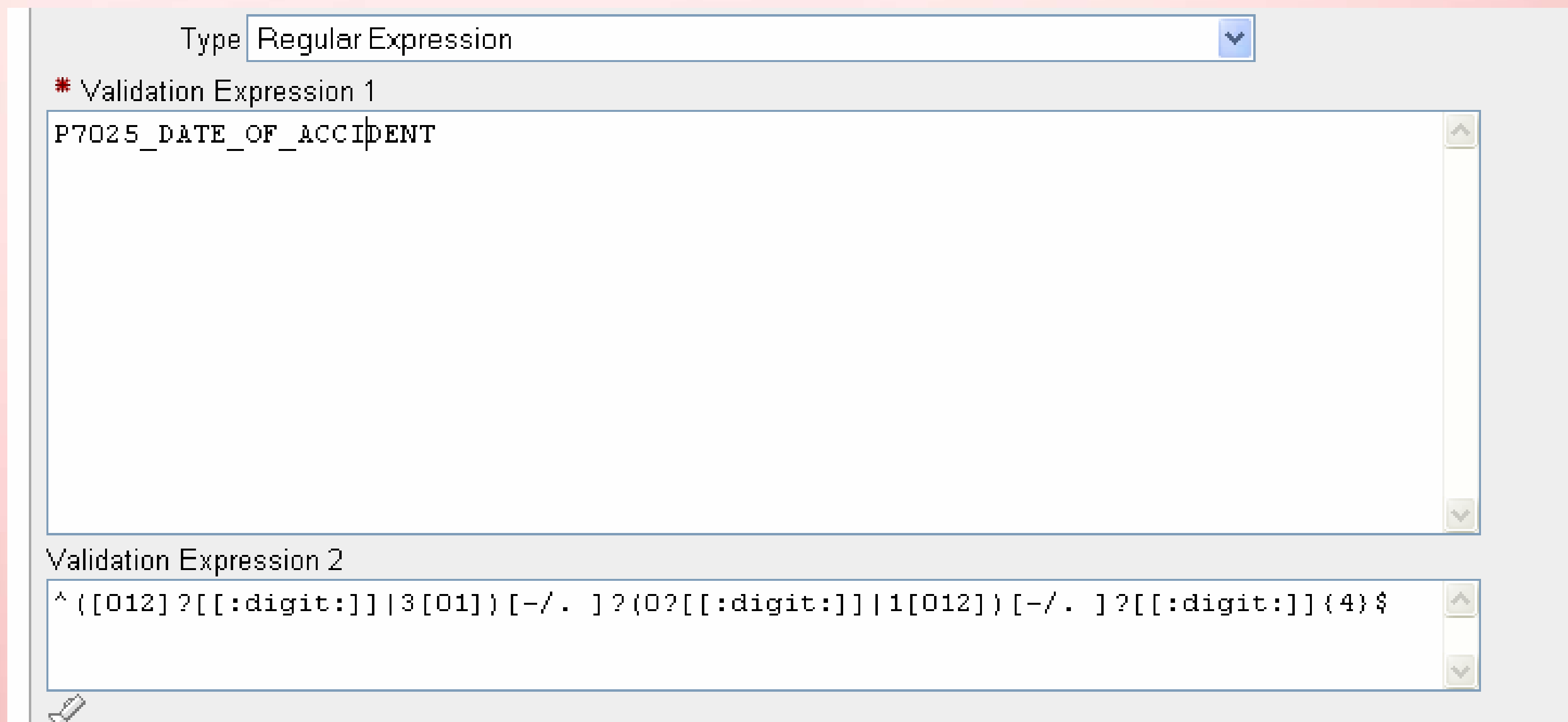
Many Text Editors use RegEx to locate text.

Unix and Regular Expressions



BURLESON CONSULTING

APEX and Regular Expressions



BURLESON CONSULTING

Regular Expression Patterns

Characters and Numbers represent themselves. ‘abc’ matches abc.

. - represents a single character or number. The pattern ‘b.e’ matches bee, bye, b6e, but not bei or b67e.

* - represents zero or more characters.

+ - represents one or more characters.

? - zero or one character.

BURLESON
CONSULTING

Regular Expression Patterns

I want to represent a US telephone number:

Will this work?

...-...-....

How about this?

__***

BURLESON
CONSULTING

Regular Expression Patterns

I want to represent a US telephone number:

Will this work?

...-...-....

123-456-7890 will match.

BURLESON
CONSULTING

Regular Expression Patterns

I want to represent a US telephone number:

Will this work?

...-...-....

123-456-7890 will match.

So will: abc-def-hijk

Not the best solution.

BURLESON
CONSULTING

Regular Expression Patterns

{count} - defines an exact number of characters.

'a{3}' defines exactly three character a's or 'aaa'.

'.{4} matches any four characters.

We could define the phone number as:

.{3}-.{3}-.{4}

BURLESON
CONSULTING

Regular Expression Patterns

{min, max} – defines a minimum and a maximum number of characters.

‘.{2,8} matches any 2 or more characters, up to 8 characters.

{min,} – defines a minimum or more characters. There is no maximum.

Regular Expression Patterns

What does the pattern sto{1,}p match?

stop	stp	stoip
stoop	stooop	stoooooooooop

BURLESON
CONSULTING

Regular Expression Patterns

[ ] – defines a subset of an expression. Any one character will match the pattern.

[1234567890] will match any number, not a letter.

The phone number pattern becomes:

[1234567890]{3}-[1234567890]{3}-[1234567890]{4}

BURLESON CONSULTING

Regular Expression Patterns

You can also define a range in the brackets.

[0-9]{3}-[0-9]{3}-[0-9]{4}

Define uppercase letters: [A-Z]

Upper or lower case: [a-zA-Z]

BURLESON
CONSULTING

Regular Expression Patterns

What does the pattern st[aeiou][A-Za-z] match?

stop	stay	string
Step	steP	steal

How about abc[3-9]?

Acb1	abc3	abcd
abc8	abc9	abc2

BURLESON CONSULTING

Regular Expression Patterns

The caret (^) in a bracket matches any characters except the characters following the caret.

st[^o]p will match:

step stip strp

But not stop.

BURLESON CONSULTING

Regular Expression Patterns

^ - the pattern will match only the beginning of the string.

$ - the pattern will match only the end of the string. Does not include CR or line feeds.

^St[a-z] matches text that starts with 'St', followed by zero or more lower case letters.

BURLESON
CONSULTING

Regular Expression Patterns

stop$ - only matches stop if it is at the last word of the line.

| or vertical bar defines a Boolean OR.

[1-9]|[a-z] matches a number or lowercase letter.

BURLESON CONSULTING

Regular Expression Patterns

Escape Character (\)

Sometimes you want to match a character that has a defined meaning.

Match a number with two decimal points.

[0-9]+.[0-9]{2}

We must tell the expression parser that we want the character period (.).

BURLESON CONSULTING

Regular Expression Patterns

Use the escape character to tell the parser the period is the character period.

[0-9]+\.[0-9]{2}

Brackets sometime need to be escaped.

Not in Oracle.

.\{3\}-.\{3\}-.\{4\}

Regular Expression Patterns

Class Operators

[:digit:]	Any digit
[:alpha:]	Any upper or lower case letter
[:lower:]	Any lower case letter
[:upper:]	Any upper case letter
[:alnum:]	Upper or lower case letter or number
[:xdigit:]	Any hex digit
[:blank:]	Space or Tab

Regular Expression Patterns

Class Operators cont.

[:space:]	Space, tab, return, LF,CR, FF
[:cntrl:]	Control Character, non printing
[:print:]	Printable Character, space
[:graph:]	Printable Character, no space
[:punct:]	Punctuation, not control character or alphanumeric

Regular Expression Patterns

Phone Number with Class Operators:

[:digit:]{3}-[:digit:]{3}-[:digit:]{4}

Being Greedy

Regular Expression parsers are greedy.

Returns the largest set of characters that matches the pattern.

Think of taking the entire string and comparing it to the pattern definition. Then giving back characters until either a match or there are no more characters.

BURLESON
CONSULTING

Being Greedy

My pattern: .*4

Zero or more characters followed by a 4.

My string is: 123423434

BURLESON
CONSULTING

Being Greedy

My pattern: .*4

Zero or more characters followed by a 4.

My string is: 123423434

The first match is the entire string.

Being Greedy

My pattern: .*4

Zero or more characters followed by a 4.

My string is: 123423434

The first match is the entire string.

BURLESON
CONSULTING

Being Greedy

My pattern: ([:digit:]{3}-){3}

3 digits followed by a dash, 3 times

My string is: 123-423-434-987-

The first match is?

BURLESON
CONSULTING

Being Greedy

My pattern: ([:digit:]{3}-){3}

3 digits followed by a dash, 3 times

My string is: 123-423-434-987-

The first match is ‘123-423-434-’**.**

Matching starts from the first character.

BURLESON
CONSULTING

Expression Grouping

Also called: Tagging or Referencing

Allows a part of the pattern to be grouped.

There can only be 9 groups.

Groups are referenced using \1-9

([a-z]+) ([a-z]+)

Matches two lower case words.

BURLESON
CONSULTING

Expression Grouping

If the matching string is ‘fast stop’ then

\1 references ‘fast and \2 references ‘stop’

\1 \2 results in ‘fast stop’

\2 \1 results in ‘stop fast’

What is this RegEx?

```
(1[012]|[1-9]):[0-5][0-9]
```

BURLESON
CONSULTING

What is this RegEx?

(1[012]|[1-9]):[0-5][0-9]

Time format. 10:30, 7:45

How about this one? [:digit:]{5}(-[:digit:]{4})?

BURLESON CONSULTING

What is this RegEx?

(1[012]|[1-9]):[0-5][0-9]

Time format. 10:30, 7:45

How about this one? [:digit:]{5}(-[:digit:]{4})?

US Zip Code

How about this one? #(9*&)@$%

BURLESON CONSULTING

What is this RegEx?

(1[012]|[1-9]):[0-5][0-9]

Time format. 10:30, 7:45

How about this one? [:digit:]{5}(-[:digit:]{4})?

US Zip Code

How about this one?   #(9*&)@$%

A cartoon cuss word, not a RegEx.

BURLESON
CONSULTING

Using RegEx with Oracle

The Java Virtual Machine in the database also implements the Java support for Regular Expression.

Oracle 10g database provides 4 functions.

They operate on the database character datatypes to include VARCHAR2, CHAR, CLOB, NVARCHAR2, NCHAR, and NCLOB.

Oracle 10g RegEx Functions

- **REGEXP_LIKE** Returns true if the pattern is matched, otherwise false.
- **REGEXP_INSTR** Returns the position of the start or end of the matching string. Returns zero if the pattern is not matched.
- **REGEXP_REPLACE** Returns a string where each matching pattern is replaced with the text specified.
- **REGEXP_SUBSTR** Returns the matching string, or NULL if no match is found.

BURLESON
CONSULTING

Oracle 10g RegEx Functions

Options for all RegEx Functions

- i = case insensitive
- c = case sensitive
- n = the period will match a new line character
- m = allows the ^ and $ to match the beginning and end of lines contained in the source. Normally these characters would match the beginning and end of the source. This is for multi-line sources.

BURLESON CONSULTING

REGEXP_LIKE

Syntax: regexp_like(source, pattern(, options));

This function can be used anywhere a Boolean result is acceptable.

```
begin
   …
   if (regexp_like(n_phone_number, .*[567]$))
   then …
   end if;
   …
end;
```

BURLESON
CONSULTING

REGEXP_LIKE

```
select
   …
   phone
from
  …
where regexp_like(phone, .*[567]$);
```

REGEXP_LIKE

Lets say we have a column that hold your OraCard credit card number.

The card number is 4 sets of 4 numbers.

XXXX XXXX XXXX XXXX

First, how can we express this as an expression?

BURLESON CONSULTING

REGEXP_LIKE

xxxx xxxx xxxx xxxx

(([0-9]{4})([[:space:]])){3}[0-9]{4}

Now we can validate the column with a check constraint.

REGEXP_LIKE

```
Create table bigtble
 …
 OraCard_num  varchar2(20) constraint
    card_ck check (regexp_like(OraCard_num,
   ’(([0-9]{4})([[:space]])){3}[0-9]{4}’)),
 …
```

BURLESON
CONSULTING

REGEXP_REPLACE

Syntax: regexp_replace(source, pattern, replace string, position, occurrence, options)

```
select
  regexp_replace('We are driving south by
      south east','south', 'north')
from dual;
```

We are driving north by north east

BURLESON
CONSULTING

REGEXP_INSTR

Syntax: regexp_instr(source, pattern, position, occurrence, begin_end, options)

The begin_end defines whether you want the position of the beginning of the occurrence or the position of the end of the occurrence. This defaults to 0 which is the beginning of the occurrence. Use 1 to get the end position.

BURLESON
CONSULTING

REGEXP_INSTR

```
select
  regexp_instr('We are driving south by
          south east','south')
from dual;
16
select
  regexp_instr('We are driving south by
  south east','south', 1, 2, 1)
from dual;
30
```

BURLESON
CONSULTING

REGEXP_INSTR

```
select
  name,
  REGEXP_SUBSTR( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}') Card
from dumbtbl
where REGEXP_INSTR( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}') > 0;

JOB CLASS             7890 2345 6543 1234
CONSUMER GROUP        1234 5678 9012 3456
SCHEDULE              3456 8909 1234 6789
Mike Hammer           5678 9023 4567 1234
```

REGEXP_SUBSTR

Syntax: regexp_substr(source, pattern, position, occurrence, options)

```
select
  regexp_substr('We are driving south by
          south east','south')
from dual;
south
```

BURLESON
CONSULTING

Warning

RegEx provides a powerful pattern matching capability. But that power comes at a price.

Using the LIKE function will normally execute faster than a RegEx function. Of course it is also very restrictive in its capability.

Test Results: 3-5 times CPU

Using Bind Variables

Build the expression normally.

Looking for area code 720.

```
select *
from test1
where regexp_like(c1,'^720');

720-743-7641
```

Using Bind Variables

Change to use bind variables

```
declare
  tstval varchar2(30);
  outval varchar2(30);
begin
  tstval:='720';
  select * into outval from test1
  where regexp_like(c1,'^'||tstval);
  dbms_output.put_line('Results: '||outval);
end;
/
```

Results: 720-743-7641

BURLESON CONSULTING

Using REGEX in Queries

We have a large varchar2 column that contains free form data that was collected from many sources. Some users have OraCard numbers in the column, but they are in different locations. All OraCards have the same format.

```
Create table dumbtbl
( name  varchar2(30),
  lots_data varchar2(2000));
```

Using REGEX in Queries

Create table dumbtbl
(name varchar2(30),
  lots_data varchar2(2000));

Find the name and card numbers.

```
select
  name,
  REGEXP_SUBSTR( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}') Card
from dumbtbl
where REGEXP_LIKE( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}');

Mike Hammer    2345 7890 4567 9012
```

BURLESON
CONSULTING

SQL Commands - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://10.210.99.42:8080/apex/f?p=4500:1003:1955438008009762::NO::: Go Links

ORACLE Database Express Edition

Home Logout Help

User: OPENWORLD

Home > SQL > SQL Commands

Autocommit Display 10 Save Run

```
select
  name,
  REGEXP_SUBSTR( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}') Card
from dumbtbl
where REGEXP_SUBSTR( lots_data,
    '(([0-9]{4})([[:space:]])){3}[0-9]{4}') = '7890 2345 6543 1234';
```

Results Explain Describe Saved SQL History

NAME	CARD
JOB CLASS	7890 2345 6543 1234

1 rows returned in 0.47 seconds CSV Export

Application Express 2.1.0.00.39

Language: en-us

Copyright © 1999, 2006, Oracle. All rights reserved.

Done Internet

ON
CONSULTING

Function Based Indexes

I can create a function based index on the OraCard numbers in lots_date.

```
create index card_idx on dumbtbl
  (REGEXP_SUBSTR
(lots_data,'(([0-9]{4})([[:space:]])){3}[0-9]{4}'));
```

Making it is one thing, getting you queries to use it is another.

User: OPENWORLD

Home > SQL > SQL Commands

Autocommit Display 10 Save Run

```
select /*+ index(DUMBTBL,CARD_INSTR_IDX) */
  name,
  REGEXP_SUBSTR( lots_data,'(([0-9]{4})([[:space:]])){3}[0-9]{4}') Card,
  REGEXP_instr( lots_data,'(([0-9]{4})([[:space:]])){3}[0-9]{4}')  Instr
from dumbtbl
where REGEXP_instr( lots_data,'(([0-9]{4})([[:space:]])){3}[0-9]{4}') > 0;
```

Results Explain Describe Saved SQL History

NAME	CARD	INSTR
JOB CLASS	7890 2345 6543 1234	11
CONSUMER GROUP	1234 5678 9012 3456	10
SCHEDULE	3456 8909 1234 6789	1
Mike Hammer	5678 9023 4567 1234	1

4 rows returned in 0.82 seconds CSV Export

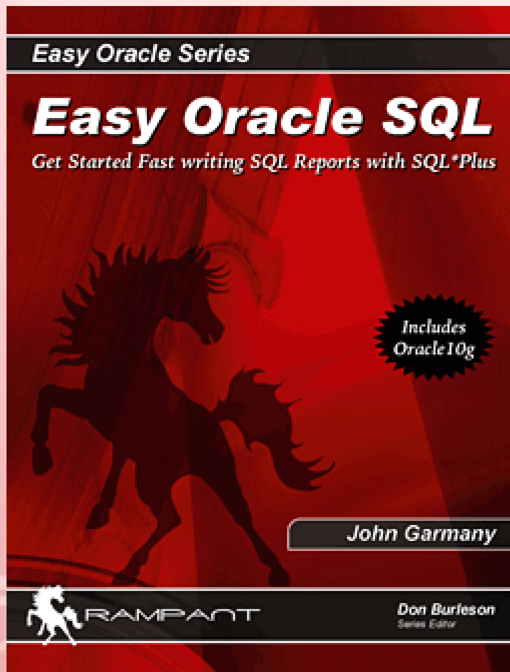
Conclusion

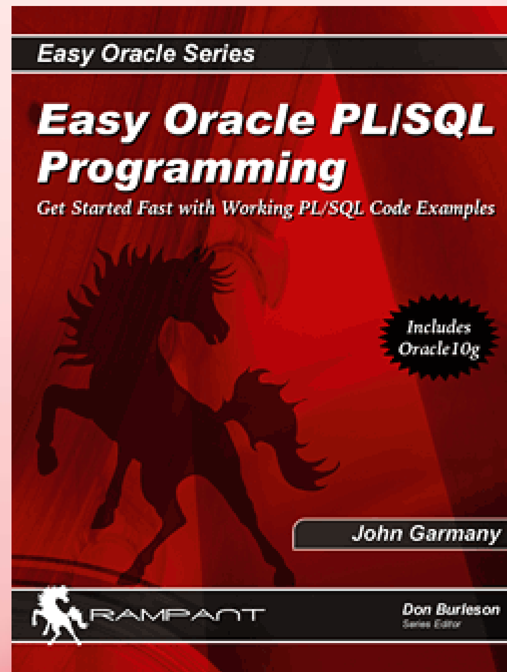
- RegEx is powerful. It can also be confusing. Verify your pattern.
- Powerful tool for data mining.
- Not always the right choice.
- Remember the Java implementation is also available.

April 15 - 19, 2007
Mandalay Bay Resort and Casino
Las Vegas, Nevada

Contact Information

John Garmany
John.garmany@computer.org





CONSULTING