Database Programming

Regular Expressions

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Describe regular expressions
- Use regular expressions to search, match, and replace strings in SQL statements
- Construct and execute regular expressions and check constraints

Purpose

Sometimes you have to find or replace a particular piece of text in a column, text string, or document.

You already know how to perform simple pattern matching using LIKE and wildcards. Sometimes you might need to look for very complex text strings such as finding the word “Winchester” in a specified text, or extracting all URLs from a piece of text. Other times you might be asked to do a more complex search such as finding all words whose every second character is a vowel.

Purpose (cont.)

Regular expressions are a method of describing both simple and complex patterns for searching and manipulating. They are used widely in the computing industry, and are not limited to Oracle. Oracle's implementation of regular expressions is an extension of the POSIX (Portable Operating System for UNIX) and are, as such, fully compatible with the POSIX standard, as controlled by the Institute of Electrical and Electronics Engineers (IEEE).

Regular Expressions

The use of regular expressions is based on the use of meta characters.

Meta characters are special characters that have a special meaning, such as a wildcard character, a repeating character, a non-matching character, or a range of characters. You can use several predefined meta character symbols in the pattern matching.

The next slides list the meta characters and provide a brief explanation of each.

META Characters

Symbol	Description
*	Matches zero or more occurrences
|	Alternation operator for specifying alternative matches
^/$	Matches the start-of-line/end-of-line
[ ]	Bracket expression for a matching list matching any one of the expressions represented in the list
{m}	Matches exactly *m* times
{m,n}	Matches at least *m* times but no more than *n* times
[: :]	Specifies a character class and matches any character in that class

META Characters (cont.)

Symbol	Description
\	Can have 4 different meanings: 1. Stands for itself 2. Quotes the next character 3. Introduces an operator 4. Does nothing
+	Matches one or more occurrences
?	Matches zero or one occurrence
.	Matches any character in the supported character set, except NULL
()	Grouping expression, treated as a single sub-expression
[==]	Specifies equivalence classes
\n	Back-reference expression
[..]	Specifies one collation element, such as a multi-character element

Regular Expression Examples

A simple regular expression is very similar to the wildcard searches you are already familiar with. Let’s take a look at an example: let’s use the dot operator to look for the letter ‘a’ followed by the letter ‘c’ with any one character separating them.

As a regular expression, this would be done as: ‘a.c’. The same expression as a standard SQL wildcard search would be: WHERE column LIKE ‘a_c’.

Regular Expression Examples (cont.)

Which of the following strings would match ‘a.c’?

‘ABC’, ‘abc’, ‘aqx’, ‘axc’, ‘aBc’, ‘abC’ ‘Amc’ ‘amrc’

Regular Expression Examples (cont.)

The strings in red would match the search string ‘a.c’

‘ABC’, ‘abc’, ‘aqx’, ‘axc’, ‘aBc’, ‘abC’ ‘Amc’ ‘amrc’

The other examples fail either due to their having the character in the wrong position or in the wrong case (uppercase not lowercase as specified in the search string).

Regular Expression Examples (cont.)

Assume you were asked to list all employees with a first name of Stephen or Steven. If you used standard Oracle wildcard searching, this would be hard to achieve, but with regular expressions, you could simply specify: '^Ste(v|ph)en$‘.

“^” specifies the start of the string that is being searched
Uppercase “S”
lowercase “t”
lowercase “e”
“(“ starts a group

Regular Expression Examples (cont.)

'^Ste(v|ph)en$‘.

lowercase “v”
“|” specifies an OR
lowercase “p”
Lowercase “h”
“)” finishes the group of choices,
lowercase “e”
lowercase “n”
“$” specifies the end of the string that is being searched

Regular Expression Functions

Oracle provides a set of SQL functions that you can use to search and manipulate strings using regular expressions.

You can use these functions on any data type that holds character data such as `CHAR`, `CLOB`, and `VARCHAR2`.

A regular expression must be enclosed in single quotation marks.

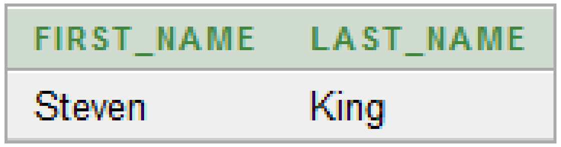
Regular Expression Functions (cont.)

Name	Description
REGEXP_LIKE	Similar to the LIKE operator, but performs regular expression matching instead of simple pattern matching
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_INSTR	Searches for a given string for a regular expression pattern and returns the position where the match is found
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and returns the matched substring
REGEXP_COUNT	Returns the number of times a pattern appears in a string. You specify the string and the pattern. You can also specify the start position and matching options (for example, c for case sensitivity).

Regular Expression Function Examples

Use of the regular expression REGEXP_LIKE could be used to solve the problem of listing either Steven or Stephen:

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```



FIRST_NAME	LAST_NAME
Steven	King

Regular Expression Function Examples (cont.)

Searching for addresses that don't start with a number and listing the position of the first non-alpha character in that address could be done like this:

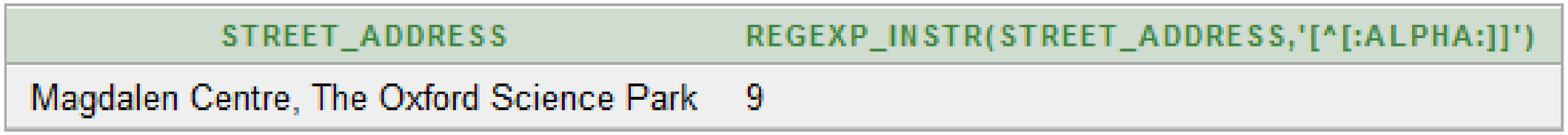
```
SELECT street_address,
               REGEXP_INSTR(street_address,'[^[:alpha:]]')
FROM     locations
WHERE  REGEXP_INSTR(street_address,'[^[:alpha:]]')> 1;
```

Explanation and result can be found on the next slide.

Regular Expression Function Examples (cont.)

```
REGEXP_INSTR(street_address,'[^[:alpha:]]')
```

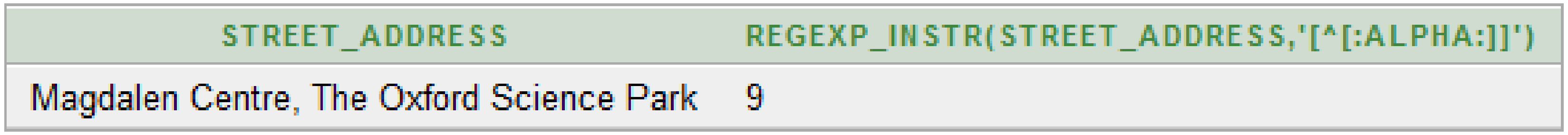
“[“ specifies the start of the expression
“^” indicates NOT when placed within the bracket
“[:alpha:]“ specifies alpha character class, i.e. not numbers
“]” ends the expression



STREET_ADDRESS	REGEXP_INSTR(STREET_ADDRESS,'[^[:ALPHA:]]')
Magdalen Centre, The Oxford Science Park	9

Regular Expression Function Examples (cont.)

In this street address, the first non-alphabetic character is the space character in the 9th character position. Those street addresses that start with a number, like 123 F. Street, have their first non-alphabetic character in the 1st character position and are excluded from the search using the WHERE clause.



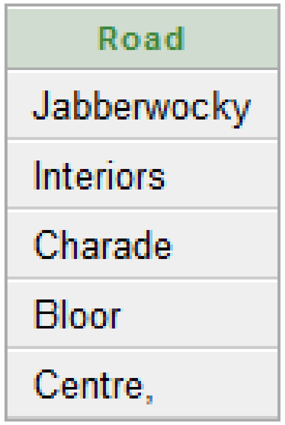
STREET_ADDRESS	REGEXP_INSTR(STREET_ADDRESS,'[^[:ALPHA:]]')
Magdalen Centre, The Oxford Science Park	9

Regular Expression Function Examples (cont.)

To return only the second word in a column containing a sentence you could issue the following statement:

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') "Road"
FROM locations;
```

This finds one or more occurrences of any non-space character [^]+ that is also preceeded by a single space and followed by a single space ' [^]+ '



Road
Jabberwocky
Interiors
Charade
Bloor
Centre,

Regular Expression Function Examples (cont.)

```
REGEXP_SUBSTR(street_address , ' [^ ]+ ')
```

"[" specifies the start of the expression
"^" indicates NOT
" " indicates a space
"]" ends the expression
"+" indicates one or more occurrences of
" " indicates a space

Regular Expression Function Examples (cont.)

Regular expressions could also be used as part of the application code to ensure that only valid data is stored in the database. It is possible to include a call to a regular expression function in, for instance, a CHECK constraint.

Regular Expression Function Examples (cont.)

So if you want to ensure that no email addresses without ‘@’ were captured in a table in your database, you could simply add the following check constraint:

```
ALTER TABLE employees
ADD CONSTRAINT email_addr_chk
CHECK(REGEXP_LIKE(email,'@'));
```

This would ensure that all email addresses include an “@” sign.

Regular Expressions in Check Constraints

Another example could be to check the format of telephone numbers:

```
CREATE TABLE contacts
      (contact_name    VARCHAR2(30),
       phone_number  VARCHAR2(30)
         CONSTRAINT c_contacts_pnf
           CHECK (REGEXP_LIKE (phone_number, '^\(\d{3}\) \d{3}-\d{4}$'))
      );
```

This constraint will ensure that all phone numbers are in the format of (XXX) XXX-XXXX.

Regular Expressions in Check Constraints (cont.)

```
(REGEXP_LIKE (phone_number, '^\(\d{3}\) \d{3}-\d{4}$'))
```

"^" The beginning of the string

"\(" A left parenthesis where at backward slash (\) is used as an escape character indicating that the left parenthesis following it is a literal rather than a grouping expression

"\d{3}" Exactly three digits

"\)" A right parenthesis. The backward slash (\) is an escape character as before

"(space character)" A space character

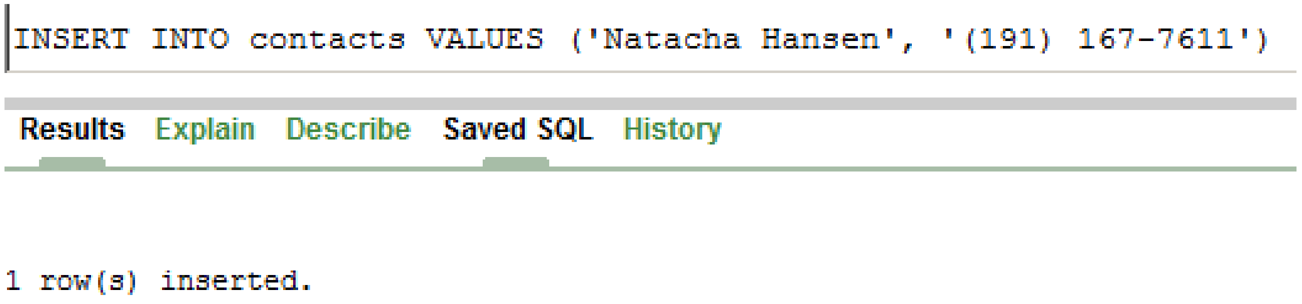
Regular Expressions in Check Constraints (cont.)

```
(REGEXP_LIKE (phone_number, '^\(\d{3}\) \d{3}-\d{4}$'))
```

“\d{3}”	Exactly three digits
“-”	A hyphen
“\d{4}”	Exactly four digits
“$”	The end of the string

Regular Expressions in Check Constraints (cont.)

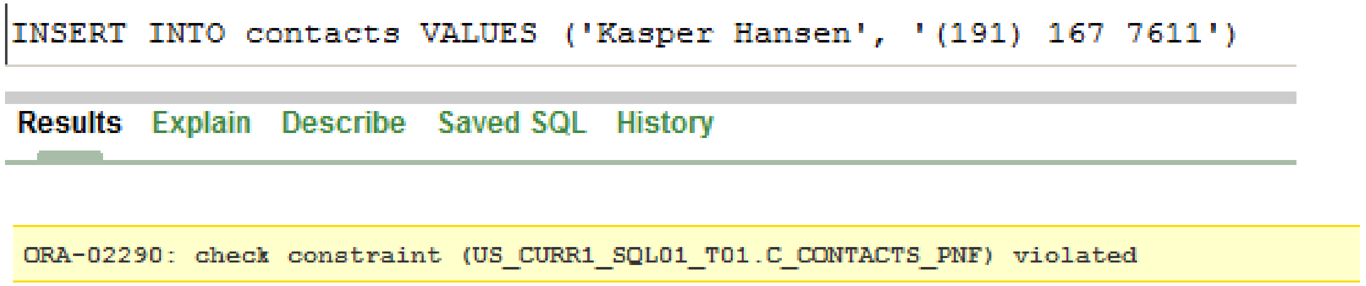
So the following row would work:



```
INSERT INTO contacts VALUES ('Natacha Hansen', '(191) 167-7611')
```

```
1 row(s) inserted.
```

And this one would not:



```
INSERT INTO contacts VALUES ('Kasper Hansen', '(191) 167 7611')
```

```
ORA-02290: check constraint (US_CURR1_SQL01_T01.C_CONTACTS_PNF) violated
```

Regular Expressions in Check Constraints (cont.)

Another use of Regular expressions in a check constraint could be to make sure a VARCHAR2 or CHAR column does not allow numbers:

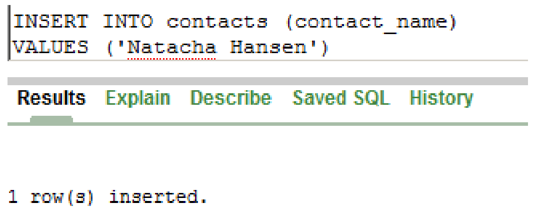
```
ALTER TABLE contacts
ADD CONSTRAINT no_number_chk
CHECK (regexp_instr(contact_name,'[[:digit:]]')=0)
```

[[:digit:]] is the POSIX expression that identified digits or numeric values. REGEXP_INSTR returns the position of any digits, and if that returned value is not = 0, then the constraint will fail.

Regular Expressions in Check Constraints (cont.)

So, the contact name ‘Natacha Hansen’ would be accepted by the database, because the number returned by regexp_instr would be 0.

regexp_instr(‘Natacha Hansen’,'[[:digit:]]‘) returns a 0, so this insert will work.

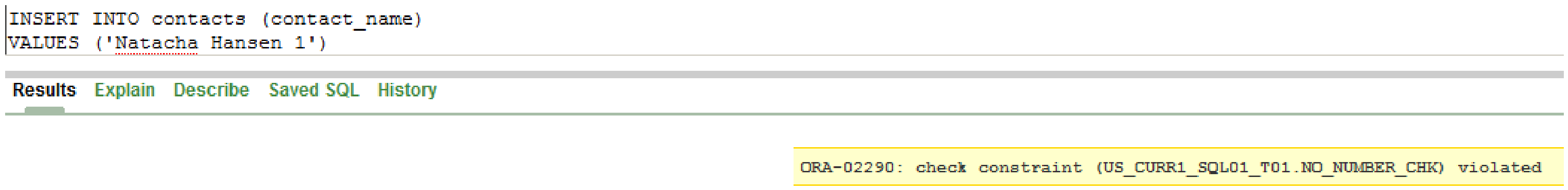


```
INSERT INTO contacts (contact_name)
VALUES ('Natacha Hansen')
```

```
1 row(s) inserted.
```

Regular Expressions in Check Constraints (cont.)

If we try to add a number to the name, for instance: ‘Natacha Hansen 1’, the insert fails. regexp_instr(‘Natacha Hansen 1’,'[[:digit:]]‘) returns 16.
16 = 0, is not true, so the insert fails.



```
INSERT INTO contacts (contact_name)
VALUES ('Natacha Hansen 1')
```

```
ORA-02290: check constraint (US_CURR1_SQL01_T01.NO_NUMBER_CHK) violated
```

Subexpressions

From Oracle 11g we can also use subexpressions when we are using regular expressions.

Parentheses are used to identify the subexpressions within the expression, and they are supported in the REGEXP_INSTR and REGEXP_SUBSTR functions.

Subexpressions (cont.)

Look at this expression:

(1 2 3) (4 (5 6) (7 8))

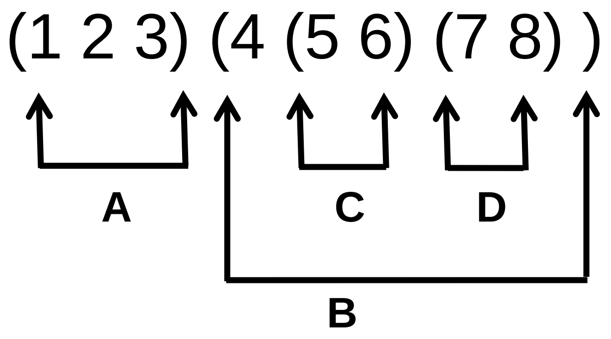
The subexpressions here are:

A. 1 2 3

B. 4 5 6 7 8

C. 5 6

D. 7 8



Subexpressions (cont.)

Subexpressions are especially useful in the real world: for instance, when working with DNA sequencing.

Look at a partial example of a mouse DNA sequence:

“ccacctttccctccactcagttctcacctgtaaagcgtccctccctcatccccatg ccccccttaccctgcagggtagagtaggctagaaaccagagagctccaagctc catctgtggagaggtgccatccttgggctgcgagagaggagaatttgcccaaa gctgcctgtttgaacgatggagacatgattgcccgtaaagggtcctgaatgcat gagatgtctttcgagagtaccggttacgggttaaaaggtcatgagacttcgatc attacgatcgtggttaacacacatatgagtatagagacacattggccaagagtt gagattgagag”

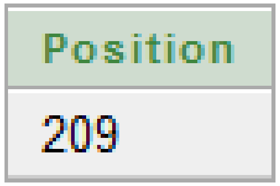
Subexpressions (cont.)

Imagine you are working with DNA sequences, and you have to find the offset (position from the start) of a specific sequence, starting with gtc followed by tcac and then aaag, in that order.

This could easily be done with the REGEXP_INSTR function, which would return the position where a match is found.

Subexpressions (cont.)

```
SELECT
REGEXP_INSTR('ccacctttccctccactcagttctcacctgtaaagcgtccctccctcatccccatgcc
cccttaccctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatcc
ttgggctgcgagagaggagaatttgcccaaagctgcctgtttgaacgatggagacatgattgcccgtaaagg
gtcctgtctcacaaaggagatgtctttcgagagtaccggttacgggttaaaaggtcatgagacttcgatcat
tacgatcgtggttaacacacatatgagtatagagacacattggccaagagttgagattgagag',
     '(gtc(tcac)(aaag))',1,1,0,'i‘,1) "Position"
FROM DUAL;
```



Position
209

Subexpressions (cont.)

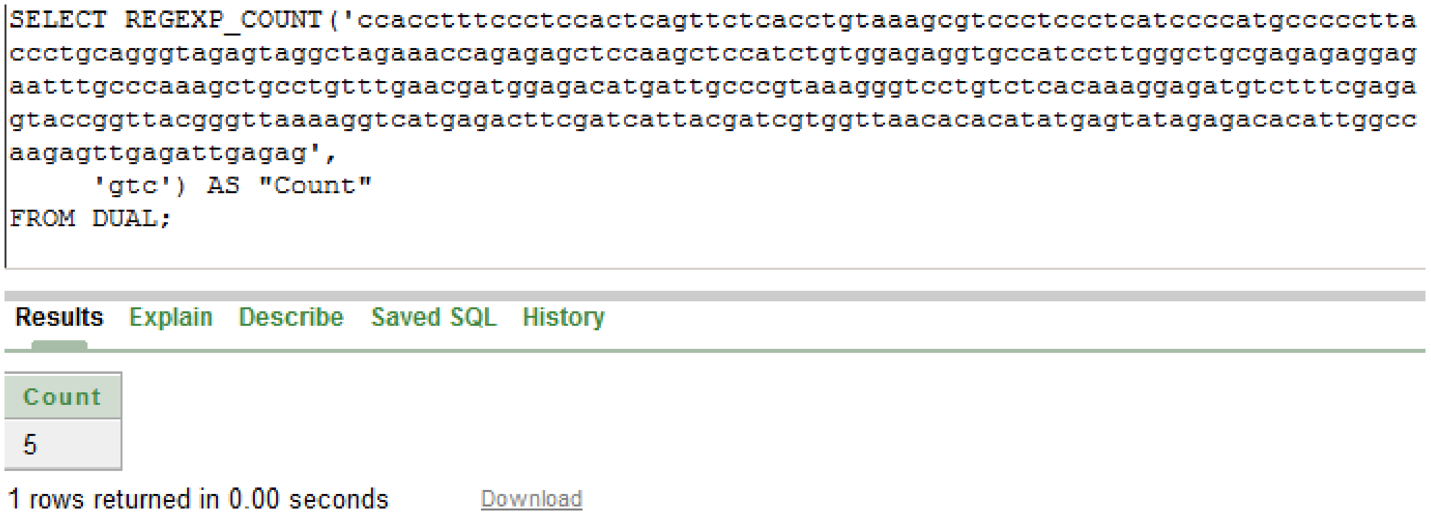
```
SELECT
REGEXP_INSTR('ccacctttccctccactcagttctcacctgtaaagcgtccctccctcatccccatgcc
cccttaccctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatcc
ttgggctgcgagagaggagaatttgcccaaagctgcctgtttgaacgatggagacatgattgcccgtaaagg
gtcctgtctcacaaaggagatgtctttcgagagtaccggttacgggttaaaaggtcatgagacttcgatcat
tacgatcgtggttaacacacatatgagtatagagacacattggccaagagttgagatt
gagag', -- The string you are searching in
     '(gtc(tcac)(aaag))', -- The subexpressions, here we have 3
         1, -- Start position of search
         1, -- Identifies occurrence of pattern you are searching for. 1
               means First occurrence
         0, -- Return option. The position of the character following
               the occurrence is returned. 0 is position of match and 1
               means the character position after the occurrence is
               returned.
         'i‘, -- Case insensitive or not
         0) "Position“ -- Which subexpression you want returned, 1, 2 or
                           3 would be valid values in this example, as we
                           have three subexpressions.
FROM DUAL;
```

REGEXP_COUNT

Oracle 11g also has a new regular expression function: REGEXP_COUNT. This function greatly simplifies counting the number of times a pattern appears inside a string.

REGEXP_COUNT (cont.)

So using the Mouse DNA example, if we wanted to count how many times the pattern ‘gtc’ was represented in the DNA sample, we could simply count them.



```
SELECT REGEXP_COUNT('ccacctttccctccactcagttctcacctgtaaagcgtccctccctcatccccatgcccctta
ccctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcgagagaggag
aatttgcccaaagctgcctgtttgaacgatggagacatgattgcccgtaaagggtcctgtctcacaaaggagatgtctttcgaga
gtaccggttacgggttaaaaggtcatgagacttcgatcattacgatcgtggttaacacacatatgagtatagagacacattggcc
aagagttgagattgagag',
     'gtc') AS "Count"
FROM DUAL;
```

```
Count
5

1 rows returned in 0.00 seconds        Download
```

Terminology

Key terms used in this lesson included:

- REGULAR EXPRESSIONS
- Subexpressions

Summary

In this lesson, you should have learned how to:

- Describe regular expressions
- Use regular expressions to search, match, and replace strings in SQL statements
- Construct and execute regular expressions and check constraints