Regular Expression Support

Objectives

After completing this appendix, you should be able to:

- List the benefits of using regular expressions
- Use regular expressions to search for, match, and replace strings

ORACLE

In this appendix, you learn to use the regular expression support feature. Regular expression support is available in both SQL and PL/SQL.

What Are Regular Expressions?

- You use regular expressions to search for (and manipulate) simple and complex patterns in string data by using standard syntax conventions.
- You use a set of SQL functions and conditions to search for and manipulate strings in SQL and PL/SQL.
- You specify a regular expression by using:
  - Metacharacters, which are operators that specify the search algorithms
  - Literals, which are the characters for which you are searching

ORACLE

Oracle Database provides support for regular expressions. The implementation complies with the Portable Operating System for UNIX (POSIX) standard, controlled by the Institute of Electrical and Electronics Engineers (IEEE), for ASCII data-matching semantics and syntax. Oracle's multilingual capabilities extend the matching capabilities of the operators beyond the POSIX standard. Regular expressions are a method of describing both simple and complex patterns for searching and manipulating.

String manipulation and searching contribute to a large percentage of the logic within a web-based application. Usage ranges from the simple, such as finding the words "San Francisco" in a specified text, to the complex task of extracting all URLs from the text and the more complex task of finding all words whose every second character is a vowel.

When coupled with native SQL, the use of regular expressions allows for very powerful search and manipulation operations on any data stored in an Oracle database. You can use this feature to easily solve problems that would otherwise involve complex programming.

Benefits of Using Regular Expressions

Regular expressions enable you to implement complex match logic in the database with the following benefits:

- By centralizing match logic in Oracle Database, you avoid intensive string processing of SQL result sets by middle-tier applications.
- Using server-side regular expressions to enforce constraints, you eliminate the need to code data validation logic on the client.
- The built-in SQL and PL/SQL regular expression functions and conditions make string manipulations more powerful and easier than in previous releases of Oracle Database.

ORACLE

Regular expressions are a powerful text-processing component of programming languages such as PERL and Java. For example, a PERL script can process each HTML file in a directory, read its contents into a scalar variable as a single string, and then use regular expressions to search for URLs in the string. One reason for many developers writing in PERL is that it has a robust pattern-matching functionality. Oracle's support of regular expressions enables developers to implement complex match logic in the database. Regular expressions were introduced in Oracle Database 10*g*.

Using the Regular Expressions Functions and Conditions in SQL and PL/SQL

Function or Condition Name	Description
REGEXP_LIKE	Is similar to the LIKE operator, but performs regular expression matching instead of simple pattern matching (condition)
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_INSTR	Searches a 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 extracts the matched substring
REGEXP_COUNT	Returns the number of times a pattern match is found in an input sting

ORACLE

I - 5

Oracle Database provides a set of SQL functions that you use to search and manipulate strings by using regular expressions. You use these functions on a text literal, bind variable, or any column that holds character data such as CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, and VARCHAR2 (but not LONG). A regular expression must be enclosed within single quotation marks. This ensures that the entire expression is interpreted by the SQL function and can improve the readability of your code.

- REGEXP_LIKE: This condition searches a character column for a pattern. Use this condition in the WHERE clause of a query to return rows matching the regular expression that you specify.
- REGEXP_REPLACE: This function searches for a pattern in a character column and replaces each occurrence of that pattern with the pattern that you specify.
- REGEXP_INSTR: This function searches a string for a given occurrence of a regular expression pattern. You specify which occurrence you want to find and the start position to search from. This function returns an integer indicating the position in the string where the match is found.
- REGEXP_SUBSTR: This function returns the actual substring matching the regular expression pattern that you specify.

- `REGEXP_COUNT`**:** This function returns the number of times a pattern match is found in the input string.

What Are Metacharacters?

- Metacharacters are special characters that have a special meaning such as a wildcard, a repeating character, a nonmatching character, or a range of characters.
- You can use several predefined metacharacter symbols in the pattern matching.
- For example, the `^(f|ht)tps?:$` regular expression searches for the following from the beginning of the string:
  - The literals `f` or `ht`
  - The `t` literal
  - The `p` literal, optionally followed by the `s` literal
  - The colon "`:`" literal at the end of the string

ORACLE

The regular expression in the slide matches the `http:`, `https:`, `ftp:`, and `ftps:` strings.

Note: For a complete list of the regular expressions' metacharacters, see the *Oracle Database Advanced Application Developer's Guide* for Oracle Database 12*c*.

Using Metacharacters with Regular Expressions

Syntax	Description
.	Matches any character in the supported character set, except NULL
+	Matches one or more occurrences
?	Matches zero or one occurrence
*	Matches zero or more occurrences of the preceding subexpression
{m}	Matches exactly *m* occurrences of the preceding expression
{m, }	Matches at least *m* occurrences of the preceding subexpression
{m,n}	Matches at least *m*, but not more than *n*, occurrences of the preceding subexpression
[...]	Matches any single character in the list within the brackets
|	Matches one of the alternatives
(...)	Treats the enclosed expression within the parentheses as a unit. The subexpression can be a string of literals or a complex expression containing operators.

ORACLE

Any character, " **.** " **:** **a.b** matches the strings **abb**, **acb**, and **adb**, but not **acc**.

One or more, " **+** " **:** **a+** matches the strings **a**, **aa**, and **aaa**, but does not match **bbb**.

Zero or one, " **?** "**:** **ab?c** matches the strings **abc** and **ac**, but does not match **abbc**.

Zero or more, " ***** "**:** **ab*c** matches the strings **ac**, **abc**, and **abbc**, but does not match **abb**.

Exact count, " **{m}** "**:** **a{3}** matches the strings **aaa**, but does not match **aa**.

At least count, " {m,} " : **a{3,}** matches the strings **aaa** and **aaaa**, but not **aa**.

Between count, " **{m,n}** " **:** **a{3,5}** matches the strings **aaa**, **aaaa**, and **aaaaa**, but not **aa**.

Matching character list, " **[...]** "**:** **[abc]** matches the first character in the strings **all**, **bill**, and **cold**, but does not match any characters in **doll**.

Or, " **|** "**:** **a|b** matches character **a** or character **b**.

Subexpression, " (...) ": **(abc)?def** matches the optional string **abc**, followed by **def**. The expression matches **abcdefghi** and **def**, but does not match **ghi.** The subexpression can be a string of literals or a complex expression containing operators.

Using Metacharacters with Regular Expressions

Syntax	Description
^	Matches the beginning of a string
$	Matches the end of a string
\	Treats the subsequent metacharacter in the expression as a literal
\n	Matches the *n*th (1–9) preceding subexpression of whatever is grouped within parentheses. The parentheses cause an expression to be remembered; a backreference refers to it.
\d	A digit character
[:class:]	Matches any character belonging to the specified POSIX character class
[^:class:]	Matches any single character *not* in the list within the brackets

Beginning/end of line anchor, " ^ " and "$": **`^def`** matches **`def`** in the string **`defghi`** but does not match **`def`** in **`abcdef`**. **`def$`** matches **`def`** in the string **`abcdef`** but does not match **`def`** in the string **`defghi`**.

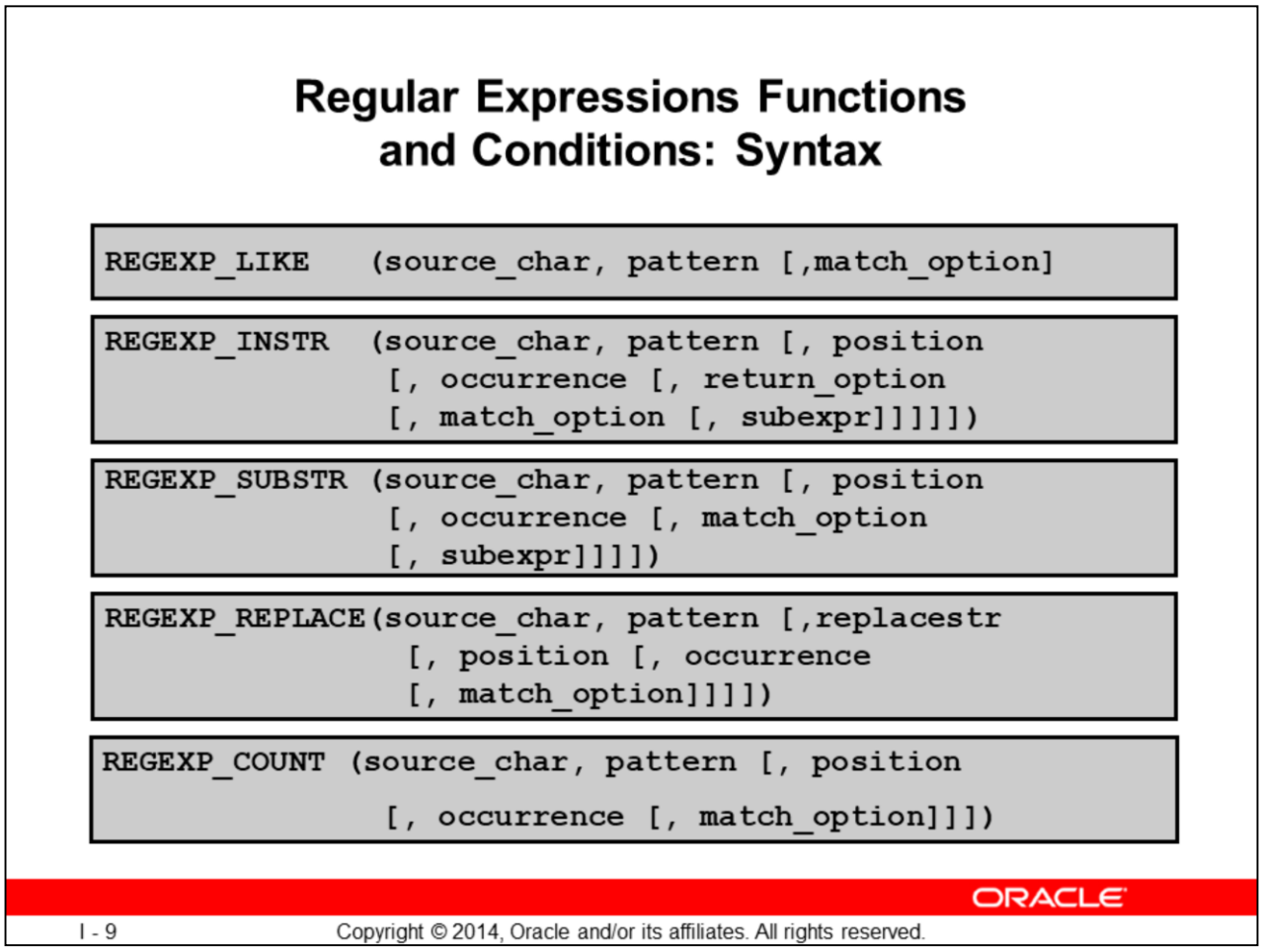
Escape character " \ ": **`\+`** searches for a **`+`**. It matches the plus character in the string **`abc+def`**, but does not match **`Abcdef`**.

Backreference, " \n ": **`(abc|def)xy\1`** matches the strings **`abcxyabc`** and **`defxydef`**, but does not match **`abcxydef`** or **`abcxy`**. A backreference enables you to search for a repeated string without knowing the actual string ahead of time. For example, the expression **`^(.*)\1$`** matches a line consisting of two adjacent instances of the same string.

Digit character, "\d": The expression `^\[\d{3}\] \d{3}-\d{4}$` matches [650] 555-1212 but does not match 650-555-1212.

Character class, " [:class:] ": **`[[:upper:]]+`** searches for one or more consecutive uppercase characters. This matches **`DEF`** in the string **`abcDEFghi`** but does not match the string **`abcdefghi`**.

Nonmatching character list (or class), " [^...] ": **`[^abc]`** matches the character **`d`** in the string **`abcdef`**, but not **`a`**, **`b`**, or **`c`**.



The syntax for the regular expressions functions and conditions is as follows:

- `source_char`: A character expression that serves as the search value
- `pattern`: A regular expression, a text literal
- `occurrence`: A positive integer indicating which occurrence of pattern in `source_char` Oracle Server should search for. The default is 1.
- `position`: A positive integer indicating the character of `source_char` where Oracle Server should begin the search. The default is 1.
- `return_option`:
  - 0: Returns the position of the first character of the occurrence (default)
  - 1: Returns the position of the character following the occurrence
- `Replacestr`: Character string replacing pattern
- `match_parameter`:
  - "`c`": Uses case-sensitive matching (default)
  - "`i`": Uses non-case-sensitive matching
  - "`n`": Allows match-any-character operator
  - "`m`": Treats source string as multiple lines

- `subexpr`: Fragment of pattern enclosed in parentheses. You learn more about subexpressions later in this appendix.

Performing a Basic Search by Using the REGEXP_LIKE Condition

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

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

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

`REGEXP_LIKE` is similar to the `LIKE` condition, except that `REGEXP_LIKE` performs regular-expression matching instead of the simple pattern matching performed by `LIKE`. This condition evaluates strings by using characters as defined by the input character set.

Example of `REGEXP_LIKE`

In this query, against the `EMPLOYEES` table, all employees with first names containing either Steven or Stephen are displayed. In the expression used `'^Ste(v|ph)en$ '`:

- `^` indicates the beginning of the expression
- `$` indicates the end of the expression
- `|` indicates either/or

Replacing Patterns by Using the REGEXP_REPLACE Function

```
REGEXP_REPLACE(source_char, pattern [,replacestr
[, position [, occurrence [, match_option]]]])
```

```
SELECT last_name, REGEXP_REPLACE(phone_number, '\.','-')
   AS phone
FROM employees;
```

Original

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

Partial results

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

ORACLE

I - 11

Using the `REGEXP_REPLACE` function, you reformat the phone number to replace the period (`.`) delimiter with a dash (`-`) delimiter. Here is an explanation of each of the elements used in the regular expression example:

- `phone_number` is the source column.
- `'\.'` is the search pattern.
  - Use single quotation marks (`' '`) to search for the literal character period (`.`).
  - Use a backslash (`\`) to search for a character that is normally treated as a metacharacter.
- `'-'` is the replace string.

Finding Patterns by Using the REGEXP_INSTR Function

```
REGEXP_INSTR  (source_char, pattern [, position [,
   occurrence [, return_option [, match_option]]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[[:alpha:]]') AS
   First_Alpha_Position
FROM  locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

ORACLE

In this example, the `REGEXP_INSTR` function is used to search the street address to find the location of the first alphabetic character, regardless of whether it is in uppercase or lowercase. Note that `[:<class>:]` implies a character class and matches any character from within that class; `[:alpha:]` matches with any alphabetic character. The partial results are displayed.

In the expression used in the query `'[[:alpha:]]'`:

- `[` starts the expression
- `[:alpha:]` indicates alphabetic character class
- `]` ends the expression

Note: The POSIX character class operator enables you to search for an expression within a character list that is a member of a specific POSIX character class. You can use this operator to search for specific formatting, such as uppercase characters, or you can search for special characters such as digits or punctuation characters. The full set of POSIX character classes is supported. Use the syntax `[:class:]`, where `class` is the name of the POSIX character class to search for. The following regular expression searches for one or more consecutive uppercase characters : `[[:upper:]]+` .

Extracting Substrings by Using the REGEXP_SUBSTR Function

```
REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option]]])
```

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

	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	Jabberwocky

In this example, the road names are extracted from the `LOCATIONS` table. To do this, the contents in the `STREET_ADDRESS` column that are after the first space are returned by using the `REGEXP_SUBSTR` function. In the expression used in the query `' [^ ]+ '`:

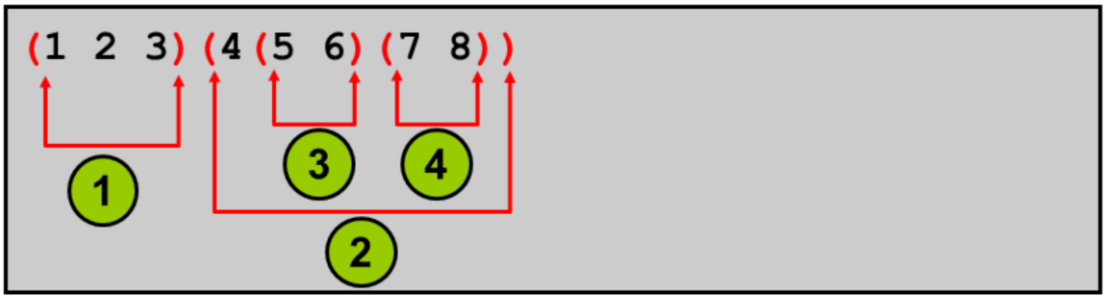
- `[` starts the expression
- `^` indicates NOT
- `' '` indicates space
- `]` ends the expression
- `+` indicates 1 or more

Subexpressions

Examine this expression:

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

The subexpressions are:



ORACLE

Oracle Database provides regular expression support parameters to access a subexpression. In the slide example, a string of digits is shown. The parentheses identify the subexpressions within the string of digits. Reading from left to right, and from outer parentheses to the inner parentheses, the subexpressions in the string of digits are:

1. 123
2. 45678
3. 56
4. 78

You can search for any of those subexpressions with the `REGEXP_INSTR` and `REGEXP_SUBSTR` functions.

Using Subexpressions with Regular Expression Support

```
SELECT
  REGEXP_INSTR
(1)('0123456789',       -- source char or search value
(2)'(123)(4(56)(78))', -- regular expression patterns
(3)1,                   -- position to start searching
(4)1,                   -- occurrence
(5)0,                   -- return option
(6)'i',                 -- match option (case insensitive)
(7)1)              -- subexpression on which to search
   "Position"
FROM dual;
```

Position
2

`REGEXP_INSTR` and `REGEXP_SUBSTR` have an optional `SUBEXPR` parameter that lets you target a particular substring of the regular expression being evaluated.

In the example shown in the slide, you may want to search for the first subexpression pattern in your list of subexpressions. The example shown identifies several parameters for the `REGEXP_INSTR` function.

1. The string you are searching is identified.
2. The subexpressions are identified. The first subexpression is `123`. The second subexpression is `45678`, the third is `56`, and the fourth is `78`.
3. The third parameter identifies from which position to start searching.
4. The fourth parameter identifies the occurrence of the pattern you want to find. `1` means find the first occurrence.
5. The fifth parameter is the return option. This is the position of the first character of the occurrence. (If you specify `1`, the position of the character following the occurrence is returned.)
6. The sixth parameter identifies whether your search should be case-sensitive or not.
7. The last parameter specifies which subexpression you want to find. In the example shown, you are searching for the first subexpression, which is `123`.

Why Access the *n*th Subexpression?

- A more realistic use: DNA sequencing
- You may need to find a specific subpattern that identifies a protein needed for immunity in mouse DNA.

```
SELECT
    REGEXP_INSTR('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctc
    cctcatccccatgccccctttaccctgcagggtagagtaggctagaaaccagagagctccaagc
    tccatctgtggagaggtgccatccttgggctgcagagagaggagaatttgccccaaagctgcc
    tgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttca
    ccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagag
    gctcttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggc
    atgtaggggcgtggggatgcgctctgctctgctctcctctcctgaacccctgaacccctctggc
    taccccagagcacttagagccag',
                    '(gtc(tcac)(aaag))',
                     1, 1, 0, 'i',
                     1) "Position"
FROM dual;
```

	Position
1	195

ORACLE

In life sciences, you may need to extract the offsets of subexpression matches from a DNA sequence for further processing. For example, you may need to find a specific protein sequence, such as the begin offset for the DNA sequence preceded by `gtc` and followed by `tcac` followed by `aaag`. To accomplish this goal, you can use the `REGEXP_INSTR` function, which returns the position where a match is found.

In the slide example, the position of the first subexpression `(gtc)` is returned. `gtc` appears starting in position 195 of the DNA string.

If you modify the slide example to search for the second subexpression `(tcac)`, the query results in the following output. `tcac` appears starting in position 198 of the DNA string.

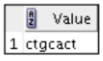
	Position
1	198

If you modify the slide example to search for the third subexpression `(aaag)`, the query results in the following output. `aaag` appears starting in position 202 of the DNA string.

	Position
1	202

REGEXP_SUBSTR: Example

```
SELECT
  REGEXP_SUBSTR
(1)('acgctgcactgca', -- source char or search value
(2) 'acg(.*)gca',    -- regular expression pattern
(3) 1,               -- position to start searching
(4) 1,               -- occurrence
(5) 'i',             -- match option (case insensitive)
(6) 1)               -- sub-expression
 "Value"
FROM dual;
```



In the example shown in the slide:

1. `acgctgcactgca` is the source to be searched.
2. `acg(.*)gca` is the pattern to be searched. Find `acg` followed by `gca` with potential characters between the `acg` and the `gca`.
3. Start searching at the first character of the source.
4. Search for the first occurrence of the pattern.
5. Use non-case-sensitive matching on the source.
6. Use a nonnegative integer value that identifies the *n*th subexpression to be targeted. This is the subexpression parameter. In this example, `1` indicates the first subexpression. You can use a value from 0–9. A zero means that no subexpression is targeted. The default value for this parameter is 0.

Using the REGEXP_COUNT Function

```
REGEXP_COUNT (source_char, pattern [, position
                [, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT(
    'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
    ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggag
    aatttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagtt
    ttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtc
    tgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctg
    ctctgctctcctctcctgaacccctgaacccctctggctaccccagagcacttagagccag',
    'gtc') AS Count
FROM dual;
```

	COUNT
1	4

The REGEXP_COUNT function evaluates strings by using characters as defined by the input character set. It returns an integer indicating the number of occurrences of the pattern. If no match is found, the function returns 0.

In the slide example, the number of occurrences for a DNA substring is determined by using the REGEXP_COUNT function.

The following example shows that the number of times the pattern 123 occurs in the string 123123123123 is three times. The search starts from the second position of the string.

```
SELECT REGEXP_COUNT
   ('123123123123', -- source char or search value
   '123',           -- regular expression pattern
    2,              -- position where the search should start
   'i')             -- match option (case insensitive)
   As Count
FROM dual;
```

	COUNT
1	3

Regular Expressions and Check Constraints: Examples

```
ALTER TABLE emp8
  ADD CONSTRAINT email_addr
  CHECK(REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
  (500,'Christian','Patel','ChrisP2creme.com',
   1234567890,'12-Jan-2004','HR_REP',2000,null,102,40);
```

```
Error starting at line 1 in command:
INSERT INTO emp8 VALUES
  (500,'Christian','Patel',
  'ChrisP2creme.com', 1234567890,
   '12-Jan-2004', 'HR_REP', 2000, null, 102, 40)
Error report:
SQL Error: ORA-02290: check constraint (TEACH_B.EMAIL_ADDR) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

ORACLE

Regular expressions can also be used in `CHECK` constraints. In this example, a `CHECK` constraint is added on the `EMAIL` column of the `EMPLOYEES` table. This ensures that only strings containing an "`@`" symbol are accepted. The constraint is tested. The `CHECK` constraint is violated because the email address does not contain the required symbol. The `NOVALIDATE` clause ensures that existing data is not checked.

For the slide example, the `emp8` table is created by using the following code:

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

Note: The example in the slide is executed by using the Execute Statement option in SQL Developer. The output format differs if you use the Run Script option.

Quiz

With the use of regular expressions in SQL and PL/SQL, you can:

a. Avoid intensive string processing of SQL result sets by middle-tier applications
b. Avoid data validation logic on the client
c. Enforce constraints on the server

ORACLE

Answer: a, b, c

Summary

In this appendix, you should have learned how to use regular expressions to search for, match, and replace strings.

ORACLE

In this appendix, you have learned to use the regular expression support features. Regular expression support is available in both SQL and PL/SQL.