11

Using Regular Expressions in Database Applications

This chapter describes regular expressions and explains how to use them in database applications.

Topics:

> **See Also:**
>
> - *Oracle Database Globalization Support Guide* for information about using SQL regular expression functions in a multilingual environment
> - *Oracle Regular Expressions Pocket Reference* by Jonathan Gennick, O'Reilly & Associates
> - *Mastering Regular Expressions* by Jeffrey E. F. Friedl, O'Reilly & Associates

11.1 Overview of Regular Expressions

A regular expression specifies a search pattern, using **metacharacters** (which are, or belong to, **operators**) and **character literals** (described in *Oracle Database SQL Language Reference*).

The search pattern can be complex. For example, this regular expression matches any string that begins with either `f` or `ht`, followed by `tp`, optionally followed by `s`, followed by the colon (`:`):

```
(f|ht)tps?:
```

The metacharacters (which are also operators) in the preceding example are the parentheses, the pipe symbol (`|`), and the question mark (`?`). The character literals are `f`, `ht`, `tp`, `s`, and the colon (`:`).

Parentheses group multiple pattern elements into a single element. The pipe symbol (`|`) indicates a choice between the elements on either side of it, `f` and `ht`. The question mark (`?`) indicates that the preceding element, `s`, is optional. Thus, the preceding regular expression matches these strings:

- `http:`

- `https:`
- `ftp:`
- `ftps:`

Regular expressions are a powerful text-processing component of the programming languages Java and PERL. For example, a PERL script can read the contents of each HTML file in a directory into a single string variable and then use a regular expression to search that string for URLs. This robust pattern-matching functionality is one reason that many application developers use PERL.

11.2 Oracle SQL Support for Regular Expressions

Oracle SQL support for regular expressions lets application developers implement complex pattern-matching logic in the database, which is useful for these reasons:

- By centralizing pattern-matching logic in the database, you avoid intensive string processing of SQL results sets by middle-tier applications.

  For example, life science customers often rely on PERL to do pattern analysis on bioinformatics data stored in huge databases of DNA and proteins. Previously, finding a match for a protein sequence such as `[AG].{4}GK[ST]` was handled in the middle tier. The SQL regular expression functions move the processing logic closer to the data, thereby providing a more efficient solution.
- By using server-side regular expressions to enforce constraints, you avoid duplicating validation logic on multiple clients.

Oracle SQL supports regular expressions with the pattern-matching condition and functions summarized in Table 11-1. Each pattern matcher searches a given string for a given pattern (described with a regular expression), and each has the pattern-matching options described in Table 11-2. The functions have additional options (for example, the character position at which to start searching the string for the pattern).

Table 11-1 Oracle SQL Pattern-Matching Condition and Functions

Name	Description
`REGEXP_LIKE`	**Condition** that can appear in the `WHERE` clause of a query, causing the query to return rows that match the given pattern. **Example:** This `WHERE` clause identifies employees with the first name of Steven or Stephen: `WHERE REGEXP_LIKE((hr.employees.first_name, '^Ste(v|ph)en$')`
`REGEXP_COUNT`	**Function** that returns the number of times the given pattern appears in the given string. **Example:** This function invocation returns the number of times that `e` (but not `E`) appears in the string `'Albert Einstein'`, starting at character position 7: `REGEXP_COUNT('Albert Einstein', 'e', 7, 'c')` (The returned value is 1, because the `c` option specifies case-sensitive matching.)

Table 11-1 (Cont.) Oracle SQL Pattern-Matching Condition and Functions

Name	Description
REGEXP_INSTR	**Function** that returns an integer that indicates the starting position of the given pattern in the given string. Alternatively, the integer can indicate the position immediately following the end of the pattern. **Example:** This function invocation returns the starting position of the first valid email address in the column `hr.employees.email`: `REGEXP_INSTR(hr.employees.email, '\w+@\w+(\.\w+)+')` If the returned value is greater than zero, then the column contains a valid email address.
REGEXP_REPLACE	**Function** that returns the string that results from replacing occurrences of the given pattern in the given string with a replacement string. **Example:** This function invocation puts a space after each character in the column `hr.countries.country_name`: `REGEXP_REPLACE(hr.countries.country_name, '(.)', '\1 ')`
REGEXP_SUBSTR	**Function** that is like `REGEXP_INSTR` except that instead of returning the starting position of the given pattern in the given string, it returns the matching substring itself. **Example:** This function invocation returns `'Oracle'` because the `x` option ignores the spaces in the pattern: `REGEXP_SUBSTR('Oracle 2010', 'O r a c l e', 1, 1, 'x')`

Table 11-2 describes the pattern-matching options that are available to each pattern matcher in Table 11-1.

Table 11-2 Oracle SQL Pattern-Matching Options for Condition and Functions

Pattern-Matching Option	Description	Example
i	Specifies case-insensitive matching.	This function invocation returns 3: `REGEXP_COUNT('Albert Einstein', 'e', 'i')`
c	Specifies case-sensitive matching.	This function invocation returns 2: `REGEXP_COUNT('Albert Einstein', 'e', 'c')`
n	Allows the Dot operator (`.`) to match the newline character, which is not the default (see Table 11-3).	In this function invocation, the string and search pattern match only because the `n` option is specified: `REGEXP_SUBSTR('a'||CHR(10)||'d', 'a.d', 1, 1, 'n')`

Table 11-2 (Cont.) Oracle SQL Pattern-Matching Options for Condition and Functions

Pattern-Matching Option	Description	Example
`m`	Specifies **multiline mode**, where a newline character inside a string terminates a line. The string can contain multiple lines. Multiline mode affects POSIX operators Beginning-of-Line Anchor (`^`) and End-of-Line Anchor (`$`) (described in Table 11-3) but not PERL-influenced operators `\A`, `\Z`, and `\z` (described in Table 11-5).	This function invocation returns `ac`: `REGEXP_SUBSTR('ab'||CHR(10)||'ac', '^a.', 1, 2, 'm')`
`x`	Ignores whitespace characters in the search pattern. By default, whitespace characters match themselves.	This function invocation returns `abcd`: `REGEXP_SUBSTR('abcd', 'a b c d', 1, 1, 'x')`

> **See Also:**
>
> *Oracle Database SQL Language Reference* fore more information about single row functions

11.3 Oracle SQL and POSIX Regular Expression Standard

Oracle SQL implementation of regular expressions conforms to these standards:

- IEEE Portable Operating System Interface (POSIX) standard draft 1003.2/D11.2

  Oracle SQL follows exactly the syntax and matching semantics for regular expression operators as defined in the POSIX standard for matching ASCII (English language) data.
- Unicode Regular Expression Guidelines of the Unicode Consortium

Oracle SQL extends regular expression support beyond the POSIX standard in these ways:

- Extends the matching capabilities for multilingual data
- Supports some commonly used PERL regular expression operators that are not included in the POSIX standard but do not conflict with it (for example, character class shortcuts and the nongreedy modifier (`?`))

> **See Also:**
>
> - POSIX Operators in Oracle SQL Regular Expressions
> - POSIX Regular Expressions Specification
> - Oracle SQL Multilingual Extensions to POSIX Standard
> - Oracle SQL PERL-Influenced Extensions to POSIX Standard

11.4 Operators in Oracle SQL Regular Expressions

Oracle SQL supports a set of common operators (composed of metacharacters) used in regular expressions.

> **Caution:**
>
> The interpretation of metacharacters differs between tools that support regular expressions. If you are porting regular expressions from another environment to Oracle Database, ensure that Oracle SQL supports their syntax and interprets them as you expect.

Topics:

11.4.1 POSIX Operators in Oracle SQL Regular Expressions

Table 11-3 summarizes the POSIX operators defined in the POSIX standard Extended Regular Expression (ERE) syntax. Oracle SQL follows the exact syntax and matching semantics for these operators as defined in the POSIX standard for matching ASCII (English language) data. Any differences in action between Oracle SQL and the POSIX standard are noted in the Description column.

Table 11-3 POSIX Operators in Oracle SQL Regular Expressions

Operator Syntax	Names	Description	Examples
.	Any Character Dot	Matches any character in the database character set, including the newline character if you specify matching option `n` (see Table 11-2). The Linux, UNIX, and Windows platforms recognize the newline character as the linefeed character (`\x0a`). The Macintosh platforms recognize the newline character as the carriage return character (`\x0d`). **Note:** In the POSIX standard, this operator matches any English character except NULL and the newline character.	The expression **`a.b`** matches the strings `abb`, `acb`, and `adb`, but does not match `acc`.
+	One or More Plus Quantifier	Matches one or more occurrences of the preceding subexpression (greedy[1]).	The expression **`a+`** matches the strings `a`, `aa`, and `aaa`, but does not match `ba` or `ab`.
*	Zero or More Star Quantifier	Matches zero or more occurrences of the preceding subexpression (greedy[1]).	The expression **`ab*c`** matches the strings `ac`, `abc`, and `abbc`, but does not match `abb` or `bbc`.

Table 11-3 (Cont.) POSIX Operators in Oracle SQL Regular Expressions

Operator Syntax	Names	Description	Examples
?	Zero or One Question Mark Quantifier	Matches zero or one occurrences of the preceding subexpression (greedy[1]).	The expression **ab?c** matches the strings abc and ac, but does not match abbc or adc.
{*m*}	Interval Exact Count	Matches exactly *m* occurrences of the preceding subexpression.	The expression **a{3}** matches the string aaa, but does not match aa.
{*m*,}	Interval At-Least Count	Matches at least *m* occurrences of the preceding subexpression (greedy[1]).	The expression **a{3,}** matches the strings aaa and aaaa, but does not match aa.
{*m*,*n*}	Interval Between Count	Matches at least *m* but not more than *n* occurrences of the preceding subexpression (greedy[1]).	The expression **a{3,5}** matches the strings aaa, aaaa, and aaaaa, but does not match aa or aaaaaa.
[*char...*]	Matching Character List	Matches any single character in the list within the brackets. In the list, all operators except these are treated as literals: • Range operator: - • POSIX character class: [: :] • POSIX collation element: [. .] • POSIX character equivalence class: [= =] A dash (-) is a literal when it occurs first or last in the list, or as an ending range point in a range expression, as in [#--]. A right bracket (]) is treated as a literal if it occurs first in the list. **Note:** In the POSIX standard, a range includes all collation elements between the start and end of the range in the linguistic definition of the current locale. Thus, ranges are linguistic rather than byte value ranges; the semantics of the range expression are independent of the character set. In Oracle Database, the linguistic range is determined by the NLS_SORT initialization parameter.	The expression **[abc]** matches the first character in the strings all, bill, and cold, but does not match any characters in doll.
[^*char...*]	Nonmatching Character List	Matches any single character *not* in the list within the brackets. For information about operators and ranges in the character list, see the description of the Matching Character List operator.	The expression **[^abc]def** matches the string xdef, but not adef, bdef, or cdef. The expression **[^a-i]x** matches the string jx, but does not match ax, fx, or ix.
[*alt1*|*alt2*]	Or	Matches either alternative.	The expression **a|b** matches the character a or b.
(*expr*)	Subexpression Grouping	Treats the expression within the parentheses as a unit. The expression can be a string or a complex expression containing operators. You can refer to a subexpression in a back reference.	The expression **(abc)?def** matches the strings abcdef and def, but does not match abcdefg or xdef.

Table 11-3 (Cont.) POSIX Operators in Oracle SQL Regular Expressions

Operator Syntax	Names	Description	Examples
`\n`	Back Reference	Matches the *n*th preceding subexpression, where `n` is an integer from 1 through 9. A back reference counts subexpressions from left to right, starting with the opening parenthesis of each preceding subexpression. The expression is invalid if fewer than *n* subexpressions precede `\n`. A back reference lets you search for a repeated string without knowing what it is. For the `REGEXP_REPLACE` function, Oracle SQL supports back references in both the regular expression pattern and the replacement string.	The expression **`(abc|def)xy\1`** matches the strings `abcxyabc` and `defxydef`, but does not match `abcxydef` or `abcxy`. The expression **`^(.*)\1$`** matches a line consisting of two adjacent instances of the same string.
`\`	Escape Character	Treats the subsequent character as a literal. A backslash (\) lets you search for a character that would otherwise be treated as a metacharacter. Use consecutive backslashes (`\\`) to match the backslash literal itself.	The expression **`abc\+def`** matches the string `abc+def`, but does not match `abcdef` or `abccdef`.
`^`	Beginning-of-Line Anchor	**Default mode:** Matches the beginning of a string. **Multiline mode:**[2] Matches the beginning of any line the source string.	The expression **`^def`** matches the substring `def` in the string `defghi` but not in the string `abcdef`.
`$`	End-of-Line Anchor	**Default mode:** Matches the end of a string. **Multiline mode:**[2] Matches the end of any line the source string.	The expression **`def$`** matches the substring `def` in the string `abcdef` but not in the string `defghi`.
`[:class:]`	POSIX Character Class	Matches any character in the specified POSIX character class (such as uppercase characters, digits, or punctuation characters). **Note:** In English regular expressions, range expressions often indicate a character class. For example, `[a-z]` indicates any lowercase character. This convention is not useful in multilingual environments, where the first and last character of a given character class might not be the same in all languages.	The expression **`[:upper:]+`**, which specifies one or more consecutive uppercase characters, matches the substring `DEF` in the string `abcDEFghi`, but does not match any substring in `abcdefghi`.
`[.element.]`	POSIX Collating Element Operator	Specifies a collating element defined in the current locale. The `NLS_SORT` initialization parameter determines the supported collation elements. This syntax lets you use a multicharacter collating element where otherwise only single-character collating elements are allowed. For example, you can ensure that the collating element `ch`, when defined in a locale such as Traditional Spanish, is treated as one character in operations that depend on the ordering of characters.	The expression **`[.ch.]`**, which specifies the collating element `ch`, matches `ch` in the string `chabc`, but does not match any substring in `cdefg`. The expression **`[a-[.ch.]]`** specifies the range from `a` through `ch`.

Table 11-3 (Cont.) POSIX Operators in Oracle SQL Regular Expressions

Operator Syntax	Names	Description	Examples
`[=char=]`	POSIX Character Equivalence Class	Matches all characters that belong to the same POSIX character equivalence class as the specified character, in the current locale. This syntax must appear within a character list; that is, it must be nested within the brackets for a character list. Character equivalents depend on how canonical rules are defined for your database locale. For details, see *Oracle Database Globalization Support Guide*.	The expression **`[[=n=]]`**, which specifies characters equivalent to `n` in a Spanish locale, matches both `N` and `ñ` in the string `El Niño`.

1 A **greedy** operator matches as many occurrences as possible while allowing the rest of the match to succeed. To make the operator nongreedy, follow it with the nongreedy modifier (`?`) (see Table 11-5).

2 Specify multiline mode with the pattern-matching option `m`, described in Table 11-2.

11.4.2 Oracle SQL Multilingual Extensions to POSIX Standard

When applied to multilingual data, Oracle SQL POSIX operators extend beyond the matching capabilities specified in the POSIX standard.

Table 11-4 shows, for each POSIX operator, which POSIX standards define its syntax and whether Oracle SQL extends its semantics for handling multilingual data. The POSIX standards are Basic Regular Expression (BRE) and Extended Regular Expression (ERE).

Table 11-4 POSIX Operators and Multilingual Operator Relationships

Operator	POSIX BRE Syntax	POSIX ERE Syntax	Multilingual Enhancement
`\`	Yes	Yes	--
`*`	Yes	Yes	--
`+`	--	Yes	--
`?`	--	Yes	--
`|`	--	Yes	--
`^`	Yes	Yes	Yes
`$`	Yes	Yes	Yes
`.`	Yes	Yes	Yes
`[ ]`	Yes	Yes	Yes
`( )`	Yes	Yes	--
`{m}`	Yes	Yes	--
`{m,}`	Yes	Yes	--
`{m,n}`	Yes	Yes	--
`\n`	Yes	Yes	Yes
`[..]`	Yes	Yes	Yes
`[::]`	Yes	Yes	Yes

Table 11-4 (Cont.) POSIX Operators and Multilingual Operator Relationships

Operator	POSIX BRE Syntax	POSIX ERE Syntax	Multilingual Enhancement
`[==]`	Yes	Yes	Yes

Multilingual data might have multibyte characters. Oracle Database lets you enter multibyte characters directly (if you have a direct input method) or use functions to compose them. You cannot use the Unicode hexadecimal encoding value of the form `\xxxx`. Oracle Database evaluates the characters based on the byte values used to encode the character, not the graphical representation of the character.

11.4.3 Oracle SQL PERL-Influenced Extensions to POSIX Standard

Oracle SQL supports some commonly used PERL regular expression operators that are not included in the POSIX standard but do not conflict with it.

Table 11-5 summarizes the PERL-influenced operators that Oracle SQL supports.

> **Caution:**
>
> PERL character class matching is based on the locale model of the operating system, whereas Oracle SQL regular expressions are based on the language-specific data of the database. In general, you cannot expect a regular expression involving locale data to produce the same results in PERL and Oracle SQL.

Table 11-5 PERL-Influenced Operators in Oracle SQL Regular Expressions

Operator Syntax	Description	Examples
`\d`	Matches a digit character. Equivalent to POSIX expression `[[:digit:]]`.	The expression **`^\(\d{3}\) \d{3}-\d{4}$`** matches `(650) 555-0100` but does not match `650-555-0100`.
`\D`	Matches a nondigit character. Equivalent to POSIX expression `[^[:digit:]]`.	The expression **`\w\d\D`** matches `b2b` and `b2_` but does not match `b22`.
`\w`	Matches a word character (that is, an alphanumeric or underscore (_) character). Equivalent to POSIX expression `[[:alnum:]_]`.	The expression **`\w+@\w+(\.\w+)+`** matches the string `jdoe@company.co.uk` but does not match `jdoe@company`.
`\W`	Matches a nonword character. Equivalent to POSIX expression `[^[:alnum:]_]`.	The expression **`\w+\W\s\w+`** matches the string `to: bill` but does not match `to bill`.
`\s`	Matches a whitespace character. Equivalent to POSIX expression `[[:space:]]`.	The expression **`\(\w\s\w\s\)`** matches the string `(a b )` but does not match `(ab)` or `(a,b.)`.
`\S`	Matches a nonwhitespace character. Equivalent to POSIX expression `[^[:space:]]`.	The expression **`\(\w\S\w\S\)`** matches the strings `(abde)` and `(a,b.)` but does not match `(a b d e)`.
`\A`	Matches the beginning of a string, in either single-line or multiline mode. Not equivalent to POSIX operator `^`.	The expression **`\AL`** matches only the first `L` in the string `Line1\nLine2\n` (where `\n` is the newline character), in either single-line or multiline mode.

Table 11-5 (Cont.) PERL-Influenced Operators in Oracle SQL Regular Expressions

Operator Syntax	Description	Examples
`\Z`	Matches the end of a string, in either single-line or multiline mode. Not equivalent to POSIX operator `$`.	The expression **`\s\Z`** matches the last space in the string `L i n e \n` (where `\n` is the newline character), in either single-line or multiline mode.
`\z`	Matches the end of a string, in either single-line or multiline mode. Not equivalent to POSIX operator `$`.	The expression **`\s\z`** matches the newline character (`\n`) in the string `L i n e \n`, in either single-line or multiline mode.
`+?`	Matches one or more occurrences of the preceding subexpression (nongreedy[1]).	The expression **`\w+?x\w`** matches **`ab`**`xc` in the string `abxcxd` (and the greedy expression **`\w+x\w`** matches **`abxc`**`xd`).
`*?`	Matches zero or more occurrences of the preceding subexpression (nongreedy[1]). Matches the empty string whenever possible.	The expression **`\w*?x\w`** matches `xa` in the string `xaxbxc` (and the greedy expression **`\w*x\w`** matches **`xaxb`**`xc`.
`??`	Matches zero or one occurrences of the preceding subexpression (nongreedy[1]). Matches the empty string whenever possible.	The expression **`a??aa`** matches `aa` in the string `aaaa` (and the greedy expression **`a?aa`** matches **`a`**`aa`).
`{m}?`	Matches exactly `m` occurrences of the preceding subexpression (nongreedy[1]).	The expression **`(a|aa){2}?`** matches `aa` in the string `aaaa` (and the greedy expression **`(a|aa){2}`** matches `aaaa`. Both the expression **`b{2}?`** and the greedy expression **`b{2}`** match `bb` in the string `bbbb`.
`{m,}?`	Matches at least `m` occurrences of the preceding subexpression (nongreedy[1]).	The expression **`a{2,}?`** matches `aa` in the string `aaaaa` (and the greedy expression **`a{2,}`** matches `aaaaa`.
`{m,n}?`	Matches at least `m` but not more than `n` occurrences of the preceding subexpression (nongreedy[1]). `{0,n}?` matches the empty string whenever possible.	The expression **`a{2,4}?`** matches `aa` in the string `aaaaa` (and the greedy expression **`a{2,4}`** matches `aaaa`.

1 A **nongreedy** operator matches as few occurrences as possible while allowing the rest of the match to succeed. To make the operator greedy, omit the nongreedy modifier (`?`).

11.5 Using Regular Expressions in SQL Statements: Scenarios

Scenarios:

- Using a Constraint to Enforce a Phone Number Format
- Example: Enforcing a Phone Number Format with Regular Expressions
- Example: Inserting Phone Numbers in Correct and Incorrect Formats
- Using Back References to Reposition Characters

11.5.1 Using a Constraint to Enforce a Phone Number Format

Regular expressions are useful for enforcing constraints—for example, to ensure that phone numbers are entered into the database in a standard format.

Table 11-6 explains the elements of the regular expression in Example: Enforcing a Phone Number Format with Regular Expressions.

Table 11-6 Explanation of the Regular Expression Elements

Regular Expression Element	Matches . . .
^	The beginning of the string.
\(	A left parenthesis. The backslash (\) is an escape character that indicates that the left parenthesis after it is a literal rather than a subexpression delimiter.
\d{3}	Exactly three digits.
\)	A right parenthesis. The backslash (\) is an escape character that indicates that the right parenthesis after it is a literal rather than a subexpression delimiter.
space character	A space character.
\d{3}	Exactly three digits.
-	A hyphen.
\d{4}	Exactly four digits.
$	The end of the string.

11.5.2 Example: Enforcing a Phone Number Format with Regular Expressions

When you create a table, you can enforce formats with regular expressions.

Example 11-1 creates a `contacts` table and adds a `CHECK` constraint to the `p_number` column to enforce this format model:

```
(XXX) XXX-XXXX
```

Example 11-1 Enforcing a Phone Number Format with Regular Expressions

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

11.5.3 Example: Inserting Phone Numbers in Correct and Incorrect Formats

The `INSERT INTO` SQL statement can be used to test how correct and incorrect formats work.

Example 11-2 shows some statements that correctly and incorrectly insert phone numbers into the `contacts` table.

Example 11-2 Inserting Phone Numbers in Correct and Incorrect Formats

These are correct:

```
INSERT INTO contacts (p_number) VALUES('(650) 555-0100');
INSERT INTO contacts (p_number) VALUES('(215) 555-0100');
```

These generate `CHECK` constraint errors:

```
INSERT INTO contacts (p_number) VALUES('650 555-0100');
INSERT INTO contacts (p_number) VALUES('650 555 0100');
INSERT INTO contacts (p_number) VALUES('650-555-0100');
INSERT INTO contacts (p_number) VALUES('(650)555-0100');
INSERT INTO contacts (p_number) VALUES(' (650) 555-0100');
```

11.5.4 Using Back References to Reposition Characters

A back reference (described in Table 11-3) stores the referenced subexpression in a temporary buffer. Therefore, you can use back references to reposition characters, as in Example 11-3. For an explanation of the elements of the regular expression in Example 11-3, see Table 11-7.

Table 11-7 explains the elements of the regular expression in Example 11-3.

Table 11-7 Explanation of the Regular Expression Elements

Regular Expression Element	Description
`^`	Matches the beginning of the string.
`$`	Matches the end of the string.
`(\S+)`	Matches one or more nonspace characters. The parentheses are not escaped so they function as a grouping expression.
`\s`	Matches a whitespace character.
`\1`	Substitutes the first subexpression, that is, the first group of parentheses in the matching pattern.
`\2`	Substitutes the second subexpression, that is, the second group of parentheses in the matching pattern.
`\3`	Substitutes the third subexpression, that is, the third group of parentheses in the matching pattern.
`,`	Inserts a comma character.

Example 11-3 Using Back References to Reposition Characters

Create table and populate it with names in different formats:

```
DROP TABLE famous_people;
CREATE TABLE famous_people (names VARCHAR2(20));
INSERT INTO famous_people (names) VALUES ('John Quincy Adams');
INSERT INTO famous_people (names) VALUES ('Harry S. Truman');
INSERT INTO famous_people (names) VALUES ('John Adams');
INSERT INTO famous_people (names) VALUES (' John Quincy Adams');
INSERT INTO famous_people (names) VALUES ('John_Quincy_Adams');
```

SQL*Plus formatting command:

```
COLUMN "names after regexp" FORMAT A20
```

For each name in the table whose format is "first middle last", use back references to reposition characters so that the format becomes "last, first middle":

```
SELECT names "names",
  REGEXP_REPLACE(names, '^(\S+)\s(\S+)\s(\S+)$', '\3, \1 \2')
    AS "names after regexp"
```

```
FROM famous_people
ORDER BY "names";
```

Result:

```
names                names after regexp
-------------------- --------------------
 John Quincy Adams    John Quincy Adams
Harry S. Truman      Truman, Harry S.
John Adams           John Adams
John Quincy Adams    Adams, John Quincy
John_Quincy_Adams    John_Quincy_Adams

5 rows selected.
```