Tipps & Tricks: April 2004

Bereich:	SQL	Erstellung:	04/2004 MP
Versionsinfo:	10.1, 10.2, 11.1	Letzte Überarbeitung:	06/2009 EF

Regular Expressions in Oracle

Mit Regulären Ausdrücken, die vor allem Perl-Programmieren und Benutzern von Editoren wie vi und Programmen wie grep bestens bekannt sind, kann man nicht nur Text nach bestimmten Zeichenmustern durchforsten, sondern auch komplizierte Textersetzungen durchführen, indem man die zu suchenden Zeichenketten durch reguläre Ausdrücke beschreibt.
Als Ergänzung zu den Funktionen LIKE, REPLACE, SUBSTR und INSTR können regular expressions in Gestalt der Funktionen REGEXP_LIKE, REGEXP_REPLACE, REGEXP_INSTR und REGEXP_SUBSTR jetzt auch in Oracle genutzt werden.
Sie erweitern die Möglichkeiten der alten Stringfunktionen um ein vielfaches.
In der Fortsetzung dieses Monatstipps (September 2008) finden Sie eine Auswahl typischer Anwendungsbeispiele für die REGEXP-Funktionen.

Hinweis:
Wegen der Komplexität sind die Regexp-Funktionen um einiges langsamer als LIKE, REPLACE, SUBSTR und INSTR. Man sollte sie also vor allem dann einsetzen, wenn

- die Suchmuster relativ komplex ist
- man pauschal nach Zeichen aus einer bestimmten Gruppe (z.B. allen Zahlen, allen Buchstaben etc.) sucht, ohne die Zeichen direkt zu kennen.
- man durch eine der neuen Funktionen einen mehrfach geschachtelten Ausdruck aus INSTR, SUBSTR etc. ersetzen kann.

Um die neuen Funktionen auszuprobieren, können Sie sich mit dem folgenden SQL-Skript eine kleine Tabelle erzeugen.

```
CREATE TABLE regexp_test (unsinn VARCHAR2(100));
 INSERT INTO regexp_test VALUES ('Telefon: ^01123A46 709 182');
 INSERT INTO regexp_test VALUES ('phone a1234568784039 ');
 INSERT INTO regexp_test VALUES ('Handy 123.456874-0847');
 INSERT INTO regexp_test VALUES ('tel 1234568740847');
 INSERT INTO regexp_test VALUES ('zu erreichen unter 123 456874 0847');
 INSERT INTO regexp_test VALUES ('123.. 45687-40847');
 INSERT INTO regexp_test VALUES ('089- 6790#407 321');
 INSERT INTO regexp_test VALUES ('Telefonnummer 54237-298 47635');
COMMIT;
```

REGEXP_LIKE

```
REGEXP_LIKE(suchstring, muster [, match_parameter ] )
```

REGEXP_LIKE prüft Zeichenketten (suchstring) auf Übereinstimmung mit dem angegebenen Muster. Die Zusammenfassung der wichtigsten Metazeichen, mit denen man die Muster erstellen kann, finden Sie am Schluss dieses Tipps

Beispiele zu REGEXP_LIKE

Alle Mitarbeiter, die entweder mit K,A, S oder M beginnen, unabhängig von Groß- oder Kleinschreibung (dafür sorgt der Parameter 'i'). Das Caret-Zeichen "^" ausserhalb der Klammer bedingt, dass nur Zeichenketten gesucht werden, die mit diesen Buchstaben anfangen. Die Liste der Zeichen wird in eckige Klammern gestellt.

```
SELECT * FROM emp WHERE REGEXP_LIKE(ename,'^[kasm]', 'i');
```

Alle Datensätze von regexp_test, die irgendwelche Zeichen enthalten, die keine Buchstaben oder Zahlen sind. Das Caret-Zeichen verkehrt die danach aufgeführten Zeichen oder Zeichenklassen in ihr Gegenteil.

```
SELECT * FROM regexp_test WHERE REGEXP_LIKE(unsinn, '[^[:alnum:]]');
SELECT * FROM regexp_test WHERE REGEXP_LIKE(unsinn, '[^A-Za-z0-9]');

 -- Kurzform \W für nicht-alphanumerische Zeichen erst ab 10.2
SELECT * FROM regexp_test WHERE REGEXP_LIKE(unsinn, '\W');
```

Alle Datensätze von regexp_test, die einen Punkt enthalten. Hier dient der Backslash zum Maskieren des Punktes, der ansonsten ein beliebiges Zeichen bedeuten würde.
SELECT * FROM regexp_test WHERE REGEXP_LIKE(unsinn, '\.');

REGEXP_REPLACE

Mit REGEXP_REPLACE können Teile des Strings ersetzt werden. Wenn der Ersatzstring leer bleibt, werden diese Teile gelöscht.

```
REGEXP_REPLACE(suchstring, muster [, ersatzstring [, position [, vorkommen [,
match_parameter ]]]])
```

position Ab dieser Position wird mit dem Ersetzen begonnen)
vorkommen 0 bedeutet, dass alle Vorkommen ersetzt werden,
<n> heißt, dass nur n-te Vorkommen ersetzt werden

Beispiele zu REGEXP_REPLACE

alle nicht numerischen Zeichen außer Minus- und Plus-Zeichen aus einem String entfernen. Innerhalb der eckigen Klammern müssen Sonderzeichen nicht durch einen Backslash maskiert werden !

```
SELECT REGEXP_REPLACE('Tel:(+49)-89-67 90 90-40','[^-+[:digit:]]') FROM dual;
--oder
SELECT REGEXP_REPLACE('Tel:(+49)-89-67 90 90-40','[^-+0-9]') FROM dual;
```

Wenn man nur einzelne Zeichenklassen ersetzen will, geht das mit den Kurzformen (ab 10.2) am besten:

```
-- alle Zeichen rauswerfen, die keine Zahl sind
SELECT REGEXP_REPLACE(unsinn, '\D') FROM regexp_test;
```

```
-- alle Zeichen rauswerfen, die eine Zahl sind
SELECT REGEXP_REPLACE(unsinn, '\d') FROM regexp_test;
```

```
-- alle alphanumerischen Zeichen rauswerfen
SELECT REGEXP_REPLACE(unsinn, '\w') FROM regexp_test;
```

```
-- alle nicht alphanumerischen Zeichen rauswerfen
SELECT REGEXP_REPLACE(unsinn, '\W') FROM regexp_test;
```

```
-- alle nicht druckbaren Zeichen entfernen (Leerzeichen, Tabzeichen, Enter)
SELECT REGEXP_REPLACE(unsinn, '\s') FROM regexp_test;
```

Umformatierung mit REGEXP_REPLACE:

Der folgende Update ersetzt zunächst alle nicht numerischen Zeichen (kursiv) und fügt dann nach je 3 Zahlen (Gruppe 1) einen Bindestrich und nach weiteren 6 Zahlen (Gruppe 2) einen Schrägstrich ein.
Durch runde Klammern markiert man die Zahlengruppen, die man umstellen will und gibt im Replace-String die gewünschte Reihenfolge und Formatierung an, wobei die Zahlengruppen durch die sogenannten Backreference (Rückwärtsreferenzen) \1, \2 ... angesprochen werden.

```
UPDATE regexp_test SET unsinn = REGEXP_REPLACE (REGEXP_REPLACE(unsinn, '\D'
), '(\d{3})(\d{6})', '\1-\2/');

SELECT * FROM regexp_test;

UNSINN
--------------------------------
011-234670/9182
123-456878/4039
123-456874/0847
123-456874/0847
123-456874/0847
123-456874/0847
089-679040/7321
542-372984/7635
```

REGEXP_SUBSTR

```
REGEXP_SUBSTR(Suchstring, muster [, position [, vorkommen [, match_parameter ]]])
```

REGEXP_SUBSTR gibt wie SUBSTR einen Teilstring zurück.
Position: Position im Suchstring, an dem der Mustervergleich anfängt, default ist 1.
In vielen Fällen kann man durch den Einsatz von REGEXP_SUBSTR eine Kombination von SUBSTR, INSTR und LENGTH ersetzen, z.B. wenn man einen längeren String wie z.B. eine Adresse in ihre Bestandteile zerlegen will. Mit den alten Stringfunktionen ist diese Ausgabe wesentlich mühsamer zu realisieren.

Beispiel zu REGEXP_SUBSTR

```
INSERT INTO regexp_test VALUES ('S. Müller, Roter Weg 13, 12345 Adorf,
Tel:123/456789');

SELECT
-- erster String ohne Komma
   REGEXP_SUBSTR(unsinn,'[^,]+', 1,1) Name,
-- zweiter String ohne Komma und Zahlen
   REGEXP_SUBSTR(unsinn,'[^,0-9]+', 1,2) Strasse,
-- erster String, der nur Zahlen enthält
   REGEXP_SUBSTR(unsinn,'[0-9]+', 1,1) Hausnummer,
```

```
-- zweiter String, der nur Zahlen enthält
    REGEXP_SUBSTR(unsinn,'[0-9]+', 1,2) PLZ,
-- fünfter String ohne Komma, Zahlen oder Leerzeichen
    REGEXP_SUBSTR(unsinn,'[^, 0-9]+', 1,5) Ort,
-- zweiter String ohne Doppelpunkt
    REGEXP_SUBSTR(unsinn,'[^:]+', 1,2) Telefon
FROM regexp_test
WHERE unsinn LIKE '%Müller%';

NAME       STRASSE      HAUSNUMMER  PLZ     ORT      TELEFON
---------- ----------- ----------  ------- -------- -----------
S. Müller  Roter Weg            13  12345   Adorf    123/456789
```

REGEXP_INSTR

```
REGEXP_INSTR (suchstring, muster, [startposition , [vorkommen , [offset ,
[match_parameter ]]]] );
```

REGEXP_INSTR ist die Funktion mit der komplexesten Syntax. Sie gibt wie INSTR die Position eines Teilstrings zurück, wird aber relativ selten gebraucht.
Der offset ist per default 0, d.h, in diesem Fall wird Startposition des Patterns zurückgegeben.
Bei einem offset von 1 wird die Position der nächsten Übereinstimmung zurückgegeben.

Beispiel zu REGEXP_INSTR

```
--Die Position der ersten Zahl in den Datensätzen der Tabelle regexp_test
SELECT REGEXP_INSTR(unsinn, '\d') position FROM regexp_test;
```

Zusammenfassung der wichtigsten Metazeichen und ihrer Bedeutung:

- Der Punkt steht für ein beliebiges Zeichen (ausser einen Zeilenumbruch).
- eckige Klammern stehen für eine Auswahl von Zeichen, [g-l] z. B. für g, h, i, j, k oder l
  [0-9A-G] für eine beliebige Ziffer oder einen Großbuchstaben von A bis G .
- Statt die Zeichenauswahl explizit anzugeben, kann man mit den sog. Charakterklassen arbeiten.
  Die wichtigsten sind:
  `[:alnum:]` alle alphanumerischen Zeichen, `[[:alnum:]]` entspricht also `[0-9A-Za-z]`
  `\w` alle alphanumerischen Zeichen und der Unterstrich "_"
  `[:alpha:]` alle Buchstaben, `[[:alpha:]]` entspricht `[A-Za-z]`
  `[:digit:]` bzw. `\d` alle Ziffern, `[[:digit:]]` entspricht `[0-9]`
  `[:space:]` bzw. `\s` alle Zeichen, die man nicht sieht, Leerzeichen, Tabzeichen, Enter...
  `\W, \D` und `S\` sind die Umkehrungen von \w, \d und \s
  Die Abkürzungen \d, \w etc. stehen seit Version 10.2 zur Verfügung.
- Das pipe-Symbol `|` bedeutet "oder", z.B. `[ae|ä], [Otto|Emma],` etc.
- Das Dollarzeichen `$` verankert das vorausgehende Zeichen am Ende einer Zeile bzw. des Suchstrings,
  `'r$'` passt z.B. auf 'Müller' oder 'Herr', aber nicht zu 'rau' oder 'Karren'
- Das Caret-Zeichen `^` hat je nachdem, wo es steht, unterschiedliche Bedeutungen
  - ein Caret-Zeichen außerhalb eckiger Klammern verankert das nachfolgende Zeichen am Beginn einer Zeile bzw. des Suchstrings
    (`'^z.*'` passt auf 'ziel', 'zeichen' etc, aber nicht auf 'platz', 'setzen')
  - ein Caret-Zeichen innerhalb eckiger Klammern schließt die nachfolgenden Zeichen aus. dementsprechend bedeutet:
    `[^[:alnum:]]` alle nicht-alphanumerischen Zeichen (Sonderzeichen, Leerzeichen...)
    `[^0-9]` alles ausser Ziffern

`[^0-9]` alles ausser Ziffern

- Der Stern `*` steht für keine oder beliebig viele Wiederholungen des vorausgehenden Elements (Zeichen oder Gruppe), '.*' passt also auf jeden String
- Das Fragezeichen `?` steht für keine oder eine Wiederholung
- Das Pluszeichen `+` steht für eine oder beliebig viele Wiederholungen
- `{n}` steht für n Wiederholungen
- `{min,max}` steht für min bis max Wiederholungen des Elements ,
  `{3,6}` z. B für 3 bis 6,
  `{3,}` für mindestens 3

Außerhalb von eckigen Klammern bzw. in Kombinationen mit Charakterklassen muss man den Metazeichen einen Backslash voranstellen, wenn man explizit nach ihnen sucht,
z.B. `\+, \?, \*` etc.

Match-Parameter für die Oracle-REGEXP-Funktionen:

- i (case insensitive)
  Groß- und Kleinschreibung wird nicht berücksichtigt
- c (case sensitive)
  Groß- und Kleinschreibung wird berücksichtigt
- n (newline)
  Der Punkt kann in diesem Fall auch für einen Zeilenumbruch stehen.
- m (multiline)
  Die Zeichenkette wird als mehrzeilige Eingabe betrachtet.^und $ können dann auf jede Zeile angewandt werden und nicht nur für Anfang und Ende des Strings.