Capgemini

ORACLE SQL

Lesson 03: Regular Expressions, Top N Analysis with Clause, Hierarchical retrieval

Lesson Objectives

To understand the following topics:

- Introduction to regular expressions
- Using metacharacters with regular expressions
- Using the regular expressions functions:
- Accessing subexpressions
- Using the REGEXP_COUNT function
- Regular expressions and check constraints
- With Clause
- Hierarchical Retrieval

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 results 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 11g.

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

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

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.

Using Meta characters 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

Performing a Basic Search by Using the REGEXP_LIKE Condition

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

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

Replacing Patterns by Using the REGEXP_REPLACE Function

Original

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

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

	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

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

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;
```

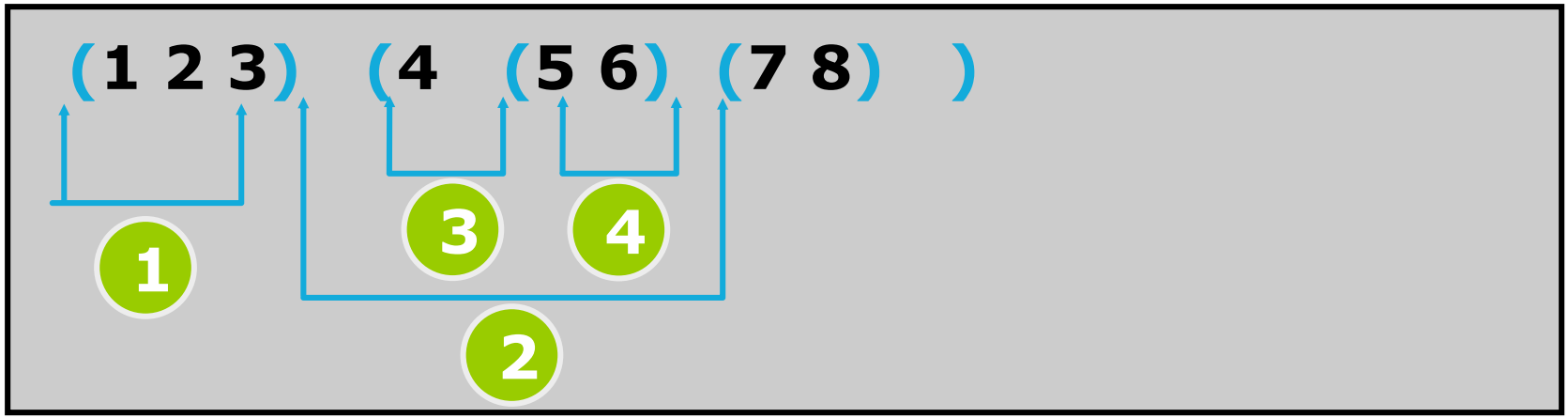
	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	Jabberwocky

Subexpressions

Examine this expression:

The subexpressions are:

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



Using Sub expressions 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)                  -- sub-expression on which to search     "Position"
FROM dual;
```

	Position
1	2

Why Access the nth Sub expression?

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('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccc
      catgccccttaccctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagag
      gtgccatccttgggctgcagagagaggagaatttgccccaaagctgcctgcagagcttcaccacccttag
      tctcacaaagccttgagttcatagcatttcttgagttttcaccctgcccagcaggacactgcagcacccaaa
      gggcttcccaggagtagggttgccctcaagaggctcttgggtctgatggccacatcctggaattgttttca
      agttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctgctctgctctcctctcctgaac
      ccctgaaccctctggctaccccagagcacttagagccag',
                  '(gtc(tcac)(aaag))',
                   1, 1, 0, 'i',
                   1) "Position"
FROM dual;
```

	Position
1	195

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;
```

Value
1 ctgcact

Using the REGEXP_COUNT Function

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

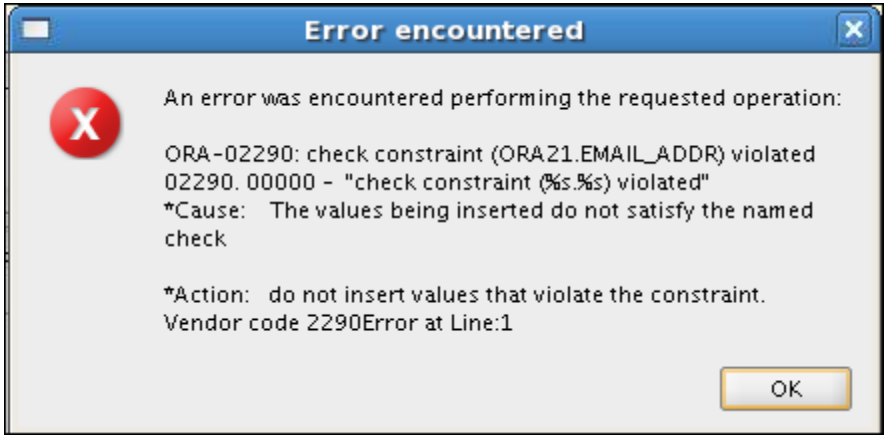
```
SELECT REGEXP_COUNT(
    'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
    ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggagaatttgc
    cccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttcaccctgcccagcagga
    cactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtctgatggccacatcctggaattgttttcaa
    gttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctgctctgctctcctctcctgaacccctgaaccctctggctac

    cccagagcacttagagccag',
    'gtc') AS Count
FROM dual;
```

	COUNT
1	4

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);
```



Row num

Top-N queries provide a method for limiting the number of rows returned from ordered sets of data.

This is very useful when you want to find the top n values in a table.

```
SELECT rownum,staff_name
FROM (SELECT staff_name,staff_sal
FROM staff_master ORDER BY staff_sal desc)
WHERE rownum < 5
```

WITH Clause

Using the WITH clause, you can use the same query block in a SELECT statement when it occurs more than once within a complex query.

The WITH clause retrieves the results of a query block and stores it in the user's temporary tablespace.

The WITH clause may improve performance.

Using WITH

WITH query_name clause allows to assign a name to a subquery block.

Subquery block can be referenced multiple places in the query by specifying the query name.

Query name is treated either an inline view or as a temporary table.

Can be specified in any top-level SELECT statement and in most types of subqueries.

The query name is visible to the main query and to all subsequent subqueries

```
WITH X_Query AS
(Select e.Ename,e.Deptno from emp e, pf p where e.Empno = p.Empno)
Select dept.Dname, X_Query.Ename from dept left outer join X_Query on dept.Deptno =
X_Query.Deptno
```

With examples

```
WITH
   dept_costs AS (
      SELECT dname, SUM(sal) dept_total
         FROM emp e, dept d
         WHERE e.deptno = d.deptno
      GROUP BY dname),
   avg_cost AS (
      SELECT avg(sal) avg
      FROM emp)
SELECT * FROM dept_costs
   WHERE dept_total >
      (SELECT avg FROM avg_cost)
      ORDER BY dname
```

Inline View instead of WITH

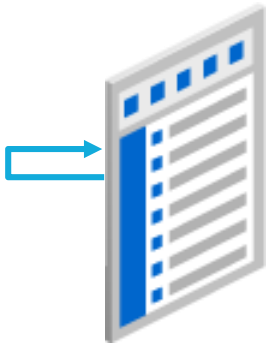
```
SELECT od.dname,dept_total, avg FROM dept od,
        ( SELECT e.deptno, dname, SUM(sal) dept_total
        FROM emp e, dept d
        WHERE e.deptno = d.deptno
      GROUP BY e.deptno,dname) dept_costs,
      (SELECT avg(sal) avg
      FROM emp) avg_cost
WHERE dept_costs.deptno = od.deptno
and  dept_total > avg
ORDER BY dname
```

Inline View

```
SELECT od.dname,dept_total, avg FROM dept od,
        ( SELECT e.deptno, dname, SUM(sal) dept_total, avg(sal) avg
        FROM emp e, dept d
        WHERE e.deptno = d.deptno
      GROUP BY e.deptno,dname
having sum(sal) > avg(sal)) dept_costs
WHERE dept_costs.deptno = od.deptno
ORDER BY dname
```

Recursive WITH Clause

The Recursive WITH clause
Enables formulation of recursive queries.
Creates query with a name, called the Recursive WITH element name
Contains two types of query blocks member: anchor and a recursive
Is ANSI-compatible



Recursive WITH Clause: Example

FLIGHTS Table

	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
   SELECT Source, Destin, Flight_time
   FROM Flights
  UNION ALL
      SELECT incoming.Source, outgoing.Destin,
            incoming.TotalFlightTime+outgoing.Flight_time
      FROM Reachable_From incoming, Flights outgoing
      WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

2

3

	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	San Jose	New York	7.1
5	Los Angeles	Boston	6.9
6	San Jose	Boston	8.2

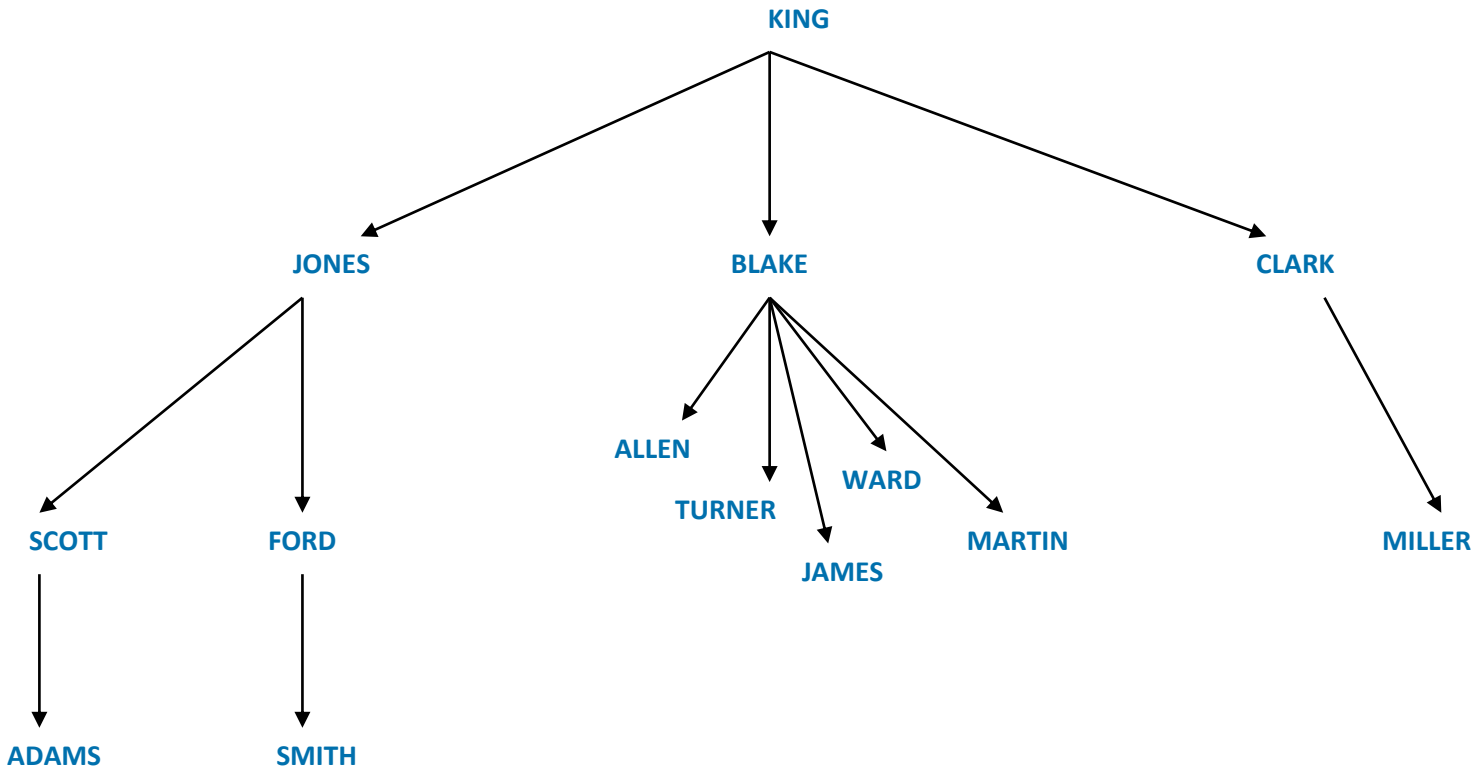
CONNECT BY and START WITH Clauses

The START WITH .. CONNECT BY clause can be used to select data that has a hierarchical relationship

- Usually, they have some sort of parent-child relationship.
- They are used to retrieve rows, which are connected to each other through a tree-like structure.

CONNECT BY and START WITH Clauses

The earliest ancestor in the tree is called the root-node called as a trunk. Extending from the trunk are branches, which have other branches.



CONNECT BY and START WITH Clauses

The restrictions on SELECT statements performing hierarchical queries are as follows:

- A SELECT statement that performs a hierarchical query cannot perform a JOIN.
- If an ORDER BY clause is used in a hierarchical query, then Oracle orders rows using the ORDER BY clause rather than in a hierarchical fashion.

CONNECT BY, START WITH Clauses-Examples

Example 1: To list “Allen” and his subordinates

```
SELECT staff_name, staff_code, mgr_code
    FROM staff_master
    CONNECT BY PRIOR staff_code = mgr_code
    START WITH staff_name = 'Allen‘;
```

- Note: If START WITH clause is omitted, then the tree structure is generated for each of the rows in the EMP table.

SUMMARY

- In this lesson, you should have learned how to use,
  - regular expressions to search for, match, and replace strings.
  - With Clause
  - Hierarchical Retrieval

Review Questions

- Question 1: ____ searches for a regular expression pattern and replaces it with a replacement string.
- Question 2: _____ met character matches the beginning of a string