

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



3.1: Regular Expressions

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.



Copyright © Capgemini 2015. All Rights Reserved 3

Benefits of Using Regular Expressions

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. This technique is useful for the following reasons:

By centralizing match logic in Oracle Database, you avoid intensive string processing of SQL results sets by middle-tier applications.

The SQL regular expression functions move the processing logic closer to the data, thereby providing a more efficient solution.

Before Oracle Database 10g, developers often coded data validation logic on the client, requiring the same validation logic to be duplicated for multiple clients. Using server-side regular expressions to enforce constraints solves this problem.

The built-in SQL and PL/SQL regular expression functions and conditions make string manipulations more powerful and less cumbersome than in previous releases of Oracle Database 10g.

3.1: Regular Expressions

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 string

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

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.

3.1: Regular Expressions

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



Copyright © Capgemini 2015. All Rights Reserved 5

What Are Metacharacters?

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 11g Release 2.

3.1: Regular Expressions

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 <i>m</i> occurrences of the preceding expression
{m, }	Matches at least <i>m</i> occurrences of the preceding subexpression
{m,n}	Matches at least <i>m</i> , but not more than <i>n</i> , 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.



Copyright © Capgemini 2015. All Rights Reserved 6

Using Metacharacters in Regular Expressions Functions

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.

3.1: Regular Expressions

Using Meta characters with Regular Expressions

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



Copyright © Capgemini 2015. All Rights Reserved 7

Using Metacharacters in Regular Expressions Functions (continued)

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**.


3.1: Regular Expressions

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

 Copyright © Capgemini 2015. All Rights Reserved 8

Performing a Basic Search by Using the REGEXP_LIKE Condition

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

3.1: Regular Expressions

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	O'Connell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

Partial results

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





Copyright © Capgemini 2015. All Rights Reserved 9

Replacing Patterns by Using the REGEXP_REPLACE Function

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.

3.1: Regular Expressions

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



Copyright © Capgemini 2015. All Rights Reserved 10

Finding Patterns by Using the REGEXP_INSTR Function

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:]]+` .


3.1: Regular Expressions

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

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

Extracting Substrings by Using the REGEXP_SUBSTR Function

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
- indicates space

3.1: Regular Expressions

Subexpressions

- Examine this expression:
`(1 2 3) (4 (5 6) (7 8))`
- The subexpressions are:

Capgemini
CONSULTING TECHNOLOGY SERVICES

Copyright © Capgemini 2015. All Rights Reserved 12

Subexpressions

Oracle Database 11g provides regular expression support parameter 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.

3.1: Regular Expressions

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

Capgemini
CONSULTING TECHNOLOGY SERVICES

Copyright © Capgemini 2015. All Rights Reserved 13

Using Subexpressions with Regular Expression Support

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 is the parameter added in Oracle Database 11g. This parameter specifies which subexpression you want to find. In the example shown, you are searching for the first subexpression, which is 123.

3.1: Regular Expressions

Why Access the *n*th 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('ccaccttccctccactcctcacgttctcacctgtaaagcgtccctccctcatcccatgcc
  ccttacccctgcaggtagagtaggctagaaccagagagctcaagctccatctgtggagaggtgccatcctgggc
  tgcagagagaggagaatttgcctcaagctgctgcagagcttcaccaccttagctcacaagccttgagttcatag
  cattcttgagtttcaccctgccagcaggacactgcagcaccctaaagggctccaggagtaggggtgccctcaaga
  ggctctgggtctgatggccacatcctggaattgtttcaagtgatggtcacagccctgaggcatgagggcggtggga
  tgcgctctgctctcctcctcctaaccctgaacctctggctacccagagacacttagagccag',
    '(gtc(tcac)(aaag))',
    1, 1, 0, 'i',
    1) "Position"
FROM dual;

```

Position
1 195



Copyright © Capgemini 2015. All Rights Reserved 14

Why Access the *n*th Subexpression?

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.

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
1 198

Position
1 202

3.1: Regular Expressions

REGEXP_SUBSTR: Example

```

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

```

Value
1 ctgcact



Copyright © Capgemini 2015. All Rights Reserved 15

REGEXP_SUBSTR: Example

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.

3.1: Regular Expressions

Using the REGEXP_COUNT Function

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

SELECT REGEXP_COUNT(

```
'ccacctttccctccactcctcagttctcactgtaaagcgtccctccctcatcccatgcccccttacctgcag
ggtagagtaggctagaaccagagagctccaagctccatctgtggagaggtgccatcctgggctgcagagagaggagaattgccccaaagctgc
ctgcagagcttcaccaccttagtctcacaagccttgagttcatagcattcttgagtttcacccctgccagcaggacactgcagcaccacaaagggctt
cccaggagtaggggtgccctcaagaggctctgggtctgatggccacatcctgggaattgtttcaagtgtatggtcacagccctgaggcatgtagggcggt
ggggatgcgctctgctctcctcctcctgaacccctgaacccctgggtacccagagacacttagagccag',
'gtc') AS Count
```

FROM dual;

	COUNT
1	4



Copyright © Capgemini 2015. All Rights Reserved 16

Using the REGEXP_COUNT Function

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 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           -- match option (case insensitive)
'i')
AS Count
FROM dual;
```

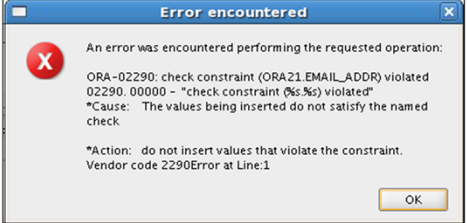
	COUNT
1	3

3.1: Regular Expressions


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);
```



The dialog box titled "Error encountered" shows an error message: "An error was encountered performing the requested operation: ORA-02290: check constraint (ORA21.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. Vendor code 2290Error at Line:1". There is an "OK" button at the bottom right.

 Capgemini
CONSULTING TECHNOLOGY ENTREPRENEUR

Copyright © Capgemini 2015. All Rights Reserved 17

Regular Expressions and Check Constraints: Examples

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.

3.2: Top N analysis

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
```

3.3: With Clause

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.



Copyright © Capgemini 2015. All Rights Reserved 19

WITH Clause

Using the WITH clause, you can define a query block before using it in a query. The WITH clause (formally known as `subquery_factoring_clause`) enables you to reuse the same query block in a SELECT statement when it occurs more than once within a complex query. This is particularly useful when a query has many references to the same query block and there are joins and aggregations.

Using the WITH clause, you can reuse the same query when it is costly to evaluate the query block and it occurs more than once within a complex query. Using the WITH clause, the Oracle server retrieves the results of a query block and stores it in the user's temporary tablespace. This can improve performance.

WITH Clause Benefits

- Makes the query easy to read

- Evaluates a clause only once, even if it appears multiple times in the query

- In most cases, may improve performance for large queries

3.3: With Clause

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

3.3: With Clause

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

3.3: With Clause

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

3.3: With Clause

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

3.3: With Clause

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



Copyright © Capgemini 2015. All Rights Reserved 24

Recursive WITH Clause

In Oracle Database 11g Release 2, the WITH clause has been extended to enable formulation of recursive queries.

Recursive WITH defines a recursive query with a name, the *Recursive WITH element name*. The Recursive WITH element definition must contain at least two query blocks: an anchor member and a recursive member.

There can be multiple anchor members but there can be only a single recursive member.

The recursive WITH clause, Oracle Database 11g Release 2 *partially* complies with the American National Standards Institute (ANSI). Recursive WITH can be used to query hierarchical data such as organization charts.

3.3: With Clause

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

	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

3

Capgemini
CONSULTING TECHNOLOGY SERVICES

Copyright © Capgemini 2015. All Rights Reserved 25

Recursive WITH Clause: Example

The example 1 in the slide displays records from a FLIGHTS table describing flights between two cities.

Using the query in example 2, you query the FLIGHTS table to display the total flight time between any source and destination. The WITH clause in the query, which is named Reachable_From, has a UNION ALL query with two branches. The first branch is the *anchor* branch, which selects all the rows from the Flights table. The second branch is the recursive branch. It joins the contents of Reachable_From to the Flights table to find other cities that can be reached, and adds these to the content of Reachable_From. The operation will finish when no more rows are found by the recursive branch.

Example 3 displays the result of the query that selects everything from the WITH clause element Reachable_From.

For details, see:

Oracle Database SQL Language Reference 11g Release 2.0

Oracle Database Data Warehousing Guide 11g Release 2.0

3.4: Hierarchical Retrieval

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.



Copyright © Capgemini 2015. All Rights Reserved 26

CONNECT BY and START WITH Clauses:

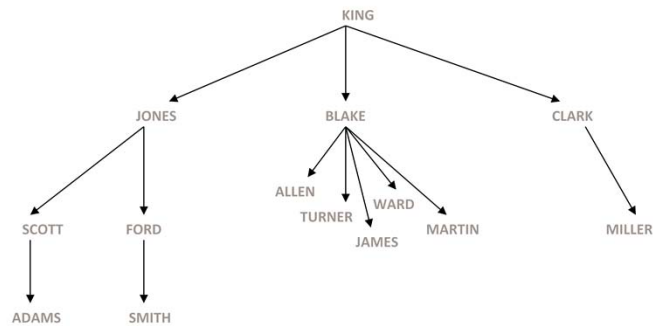
A pseudocolumn behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values.

For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on. A root row is the highest row within an inverted tree. A child row is any nonroot row. A parent row is any row that has children. A leaf row is any row without children

3.4: Hierarchical Retrieval

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.



3.4: Hierarchical Retrieval

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.

3.4: Hierarchical Retrieval

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



Copyright © Capgemini 2015. All Rights Reserved 30

Summary

This lesson addressed some of the datetime functions available in the Oracle database.

Review Question

- 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



Add the notes here.