An Illustrative Approach to Use SQL Functions: A Review

Kamalinder Kaur
Assistant Professor
Chandigarh Engineering College
Punjab, India

Abstract—This paper describes the function used in databases for performing the calculations, modifies the data items and does manipulations on data. It defines the functions, their syntax's and errors occurred during the process. Functions are applied on oracle having SQL and are illustrated through query processing.

Keywords- SQL; NVL; NULLIF; CASE; NVL2; DECODE.

I. INTRODUCTION
A SQL functions are brought up into oracle databases and are obtainable for its utilization in SQL queries.

II. BENEFITS OF SQL FUNCTIONS
The feature of SQL is its SQL Functions. These functions perform below tasks:
1) Executing calculations on data
2) Modification of individual data elements
3) Manipulate the results for collection of rows
4) Changing date and numbers to display
5) Conversion of data types of column

III. TYPES OF SQL FUNCTIONS
Single Row Function: These functions are applied on individual rows and then gives output on single row basis. The kinds of single-row functions are:
- On Character: Accepts character input and gives back both character and number digits.
- On Number: Accepts character input and gives back both character and numerical values.
- On Date: It works on values of the DATE data type. Almost all date functions outputs a value of DATE data type but the MONTHS_BETWEEN gives a number.
- Conversion: alters value from one data type to another
- COALESCE, NVL, NULLIF, CASE, NVL2, DECODE are common functions.

Character Functions
Character cases handling functions: (Lower, Initcapand Upper)

<table>
<thead>
<tr>
<th>Table 1: Character Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>LOWER('Hello Word')</td>
</tr>
<tr>
<td>UPPER('Hello Word')</td>
</tr>
<tr>
<td>INITCAP('Hello Word')</td>
</tr>
</tbody>
</table>

IV. EXECUTION OF QUERIES

This can work along with where clause: Select ename"Emp_Name",upper(job) "Job", hiredate from emp1 where ename='Anderson';

No rows selected.

This can work along with where clause: Select ename"Emp_Name",upper(job) "Job", hiredate from emp1 where ename='Anderson';

No rows selected.

Query with initcap,lower,upper function with where clause:
select ename "Emp_Name", upper(job) "Job", hiredate from emp1 where initcap (ename) ="Anderson";

Character-Manipulation Functions:
CONCAT: Joins the strings.
SUBSTR: Extracts the measurement lengthwise of the sub string.
LENGTH: It shows how long the string is numerically.
INSTR: helps in finding the numbered position of anyalphabet which is used.
LPAD: It justified the characters along the right position.
RPAD: It justified the characters along the left position.
TRIM: It cuts the front and last characters out of a string.

TABLE 2: Character – Manipulation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concat('Data', 'Structure')</td>
<td>DataStructure</td>
</tr>
<tr>
<td>Substr(&quot;Data Structure&quot;,1,4)</td>
<td>Data</td>
</tr>
<tr>
<td>Length(&quot;Data Structure&quot;)</td>
<td>13</td>
</tr>
<tr>
<td>Instr('DataStructure', 'S')</td>
<td>5</td>
</tr>
<tr>
<td>Lpad(salary,5,'*')</td>
<td>**240</td>
</tr>
<tr>
<td>Rpad(salary, 5, '*')</td>
<td>240**</td>
</tr>
<tr>
<td>Replace('BACK and BUE', 'B', 'BL')</td>
<td>BLACK and BLUE</td>
</tr>
<tr>
<td>Trim('D' from 'DataStructure')</td>
<td>ataStructure</td>
</tr>
</tbody>
</table>

Select ename "Name", job "Desg.", concat(ename, job) "Concate Fun." from emp1 where empno in (1',2',3',4');

use || symbol for cancat:
select ename "Name" , job "Desg.", ename || ' is ' || job "Concate Fun." from emp1 where empno in (1',2',3',4')

Example of Length and instr

Example of SUBSTR,LPAD,RPAD

Select ename,substr(ename,1,4),sal,lpad(sal,10,'#'),rpad(sal,10,'#')from emp1
Example of Substr and Replace

select ename, substr(ename,1,3), replace(ename, 'a','u') from emp1 where ename like '%a%';

Figure 8: Example of Substr and Replace

SQL statement displays the data for those employees whose last names end with the letter n.

Select ename, substr(ename,1,4), length(ename), instr(ename,'n') from emp1 where SUBSTR(ename, -1, 1) = 'n';

Figure 9: last names end with the letter n.

Number Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND(column</td>
<td>expression, n)</td>
</tr>
<tr>
<td>TRUNC(column</td>
<td>expression, n)</td>
</tr>
</tbody>
</table>

Select round(45.923,2), round(45.923,1), round(45.923,-1), round(44.923,-1) from dual;

Figure 10: Round function.

Select trunc(45.923,2), trunc(45.923,1), trunc(45.923,-1), trunc(44.923,-1) from dual;

Figure 11: Trunc function

select ename "Emp_Name", sal "Sal.", MOD(sal, 1000) from emp1 where empno in ('1','2','3','4');

Figure 12: Mod function

Operating Dates:
The Oracle records dates in an interior syntax:
Century-year-month-day-hours-minutes-seconds.
The automatic date demonstrates syntax is DD-MON-YY

select ename, hiredate from emp1;
HIREDATE results as DD-MON-YY. This data is stored internally as follows:

<table>
<thead>
<tr>
<th>Cent</th>
<th>Year</th>
<th>Month</th>
<th>Day</th>
<th>Hour</th>
<th>Minute</th>
<th>Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>12</td>
<td>12</td>
<td>17</td>
<td>17</td>
<td>10</td>
<td>43</td>
</tr>
</tbody>
</table>

Calculation on Dates:

1) The resultant date value can be added or subtracted to or from a date.
2) The no. of days can be calculated between two by subtracting them.
3) Hours to a date can be calculated by dividing the number of hours by 24.

```sql
SELECT ename, hiredate, hiredate+7 AS "7days+hiredate", hiredate-7 AS "7days-hiredate" FROM emp1;
```

Features of Date:

1) **Months_Between**(date1, date2):
   - It helps us in finding the no. of months between two dates. If date1 is after date2, output is positive; if date1 is earlier than date2, the output is negative.
   - The non-integer portion of the output shows a segment of the month.

2) **Add_Months**(date, n): Add no. of months into the existing calendar date. It works only on integer values and also can be negative.

3) **Next_Day**(date, 'char'): Locates the next day date after the given date. It gives output in character.

4) **Last_Day**(date): Discovers the end date of the month while considering the given date.

5) **ROUND**(date [, 'fmt']): Yields rounding of the date to specified syntax. If the syntax fmt is neglected, then date is rounded of the nearby date.

6) **TRUNC**(date [, 'fmt']): It yields the date after the time truncated from it. If the syntax fmt is neglected, then date is truncated to the nearby date.

```sql
SELECT ename, hiredate, relievingdate, round(Months_Between(relievingdate, hiredate), 0) AS "Ex._Month", Add_Months(hiredate, Next_Day(hiredate, 'SUNDAY'), Last_Day(hiredate)) FROM emp1;
```
In Where Clause

Select ename, hiredate, relievingdate, round(Months_Between (relievingdate, hiredate), 0) "Exp. Month" from emp1
Where Months_Between (relievingdate, hiredate) >= 12

Round and Truncate Function with Dates

Select ename, hiredate, relievingdate, round(hiredate, 'MONTH'),
TRUNC(hiredate, 'MONTH'), ROUND(hiredate, 'YEAR'),
TRUNC(hiredate, 'YEAR'), ROUND(hiredate, 'DAY'),
TRUNC(hiredate, 'DAY') FROM EMP1;

Conversion Functions

If Oracle server needs to convert one data type to the other then it can repeatedly converts the data to expected data type. The expected data type by the Oracle server conversion can occur wholly and clearly by the user. For this purpose some functions are required to forcefully convert the data casting to another known as conversion functions. The function names follow the conventional input data type TO output data type.

1) Conversion Type: Implicit Data Type

CHAR, VARCHAR2 can be wholly changed to NUMBER or DATE. NUMBER type value can be routinely converted to character data by Oracle server. It occurs only when the character signifies a valid number or date type value correspondingly.

For example: the select queries outputs same because Oracle inside allows 1000 and '1000' as same.
Query-1
SELECT ENAME, JOB, SAL
FROM EMP1
WHERE SAL > 15000;
Query-2
SELECT ENAME, JOB, SAL
FROM EMP1
WHERE SAL > '15000';

2) Conversion: Explicit Data Type

These functions are for single row which are skillful of converting column value, literal or an expression.

TO_DATE
TO_NUMBER
TO_CHAR

3) Function: TO_CHAR

It is required to cast a numeric input value to character type using a fixed model.

Format:
TO_CHAR(num1,[format],[nls_parameter])
Think about the below SELECT query. The query syntax the HIRE_DATE and SALARY columns of EMPLOYEES table using TO_CHAR() .

SELECT ENAME, TO_CHAR (hiredate, 'MONTH DD, YYYY') HIREDATE, TO_CHAR (sal, '$999999.99') Salary FROM emp1
5) Function: TO_NUMBER

It converts a numeric datatype from a character datatype.

Syntax:

```sql
TO_NUMBER(string1, [format], [nls_parameter])
```

list of layout models which can be used to typecast character values as number using TO_NUMBER.

<table>
<thead>
<tr>
<th>Layout Model</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Denotes Century</td>
</tr>
<tr>
<td>SCC</td>
<td>It gives Century Before Christ started with -</td>
</tr>
<tr>
<td>YYYY</td>
<td>It displays Year having four numbers</td>
</tr>
<tr>
<td>SYYY</td>
<td>It gives Year before Christ with prefixed with -</td>
</tr>
<tr>
<td>IYYY</td>
<td>Gives ISO Year having four numbers</td>
</tr>
<tr>
<td>YY</td>
<td>It is Year having 2 digits</td>
</tr>
<tr>
<td>YEAR</td>
<td>Gives Year in alphabets</td>
</tr>
<tr>
<td>SYEAR</td>
<td>Yields Year in alphabets, BC prefixed with -</td>
</tr>
<tr>
<td>MONTH</td>
<td>Gives Month in alphabets (i.e. January)</td>
</tr>
<tr>
<td>MON</td>
<td>Results JAN, FEB</td>
</tr>
<tr>
<td>WW</td>
<td>Gives Week number (i.e. 1)</td>
</tr>
<tr>
<td>W</td>
<td>Gives Week digit of the month (i.e. 5)</td>
</tr>
<tr>
<td>IW</td>
<td>Gives Week digit of the year in ISO standard.</td>
</tr>
<tr>
<td>DDD</td>
<td>Results Day of years in numbers (i.e. 365)</td>
</tr>
<tr>
<td>DD</td>
<td>Results month day in values (i.e. 28)</td>
</tr>
<tr>
<td>D</td>
<td>Gives week day in numbers (i.e. 7)</td>
</tr>
</tbody>
</table>

Note: Symbols like .(comma) and .(period) are not supported in the layout models.
**Layout Model**

<table>
<thead>
<tr>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>It spelled out Year</td>
</tr>
<tr>
<td>YYYY</td>
<td>Gives Last 3, 2, or 1 digit(s) of ISO year.</td>
</tr>
<tr>
<td>IYY, IY, I</td>
<td>Gives Last 3, 2, or 1 digit(s) of ISO year.</td>
</tr>
<tr>
<td>IYY</td>
<td>Four -digit year based on the ISO standard</td>
</tr>
<tr>
<td>Q</td>
<td>It gives Quarter of year (1, 2, 3, 4; JAN-MAR = 1).</td>
</tr>
<tr>
<td>MM</td>
<td>Returns Month (01-12; JAN = 01).</td>
</tr>
<tr>
<td>MON</td>
<td>Gives name of month.</td>
</tr>
<tr>
<td>MONTH</td>
<td>Results Name of month, covering with blanks up to 9 characters.</td>
</tr>
<tr>
<td>RM</td>
<td>Gives Roman numerals for month starting from I-IX.</td>
</tr>
<tr>
<td>WW</td>
<td>Returns Week of year (1-53)</td>
</tr>
<tr>
<td>W</td>
<td>Gives Week of month (1-5)</td>
</tr>
<tr>
<td>IW</td>
<td>On the basis of ISO standard week of year is 1-52 or 1-53</td>
</tr>
<tr>
<td>D</td>
<td>Returns the week day.</td>
</tr>
<tr>
<td>DAY</td>
<td>Gives Name of day of week.</td>
</tr>
<tr>
<td>DD</td>
<td>Gives month day (1-31).</td>
</tr>
<tr>
<td>DDD</td>
<td>Gives year day (1-366).</td>
</tr>
<tr>
<td>DY</td>
<td>Name of day is abbreviated</td>
</tr>
<tr>
<td>J</td>
<td>Returns Julian day;</td>
</tr>
<tr>
<td>HH12</td>
<td>Gives days hours (1-12).</td>
</tr>
<tr>
<td>HH24</td>
<td>Gives day hour (0-23).</td>
</tr>
<tr>
<td>MLSS</td>
<td>Gives Minute (0-59).</td>
</tr>
<tr>
<td>FF</td>
<td>Returns seconds in fraction.</td>
</tr>
<tr>
<td>AM, PM</td>
<td>Gives indicator Prime Meridian</td>
</tr>
<tr>
<td>TZH, TZM, TZR</td>
<td>Results Time zone in hour, minute.</td>
</tr>
</tbody>
</table>

Example: a character string transforms into a date syntax.

```sql
SELECT TO_DATE('February 15, 1970, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.', 'NLS_DATE_LANGUAGE = American') FROM DUAL;

TO_DATE(15-FEB-70)
```

Common Functions

These are used to hold void values in database. The purpose of the common NULL controlling function is to swap the void values with a substitute value.

**NVL**

The NVL -deputies another value for a void value. NVL function can be used with all kinds of data types.

Syntax:

NVL( Arg1, replace_with )

This case includes both the constraints which are mandatory.

The SELECT statement will display 'n/a' if an employee has not been assigned any job yet i.e. JOB_ID is NULL. Else, it would exhibit the actual JOB_ID value.

```sql
SELECT first_name, NVL(JOB_ID, 'n/a') FROM employees;

NVL2

It is an improvement over the earlier NVL, Oracle presented a facility to standby data not only for NULL columns values but also for NOT NULL columns. NVL2 can be used an alternate for Null (Void) and also for non-null value.

Syntax:

NVL2( string1, value_if_NOT_null, value_if_null )

The SELECT statement under would display 'all' if the JOB_CODE for an employee is NULL. Finally, not null value of JOB CODE, it would rather display constant value 'Job done'.

```
SQL> SELECT NVL2(all, 'Job done', 'Bench') FROM employees;
```

**NULLIF**

The NULLIF is related to two arguments expr1 and expr2. If expr1 equals to expr2 then it gives NULL otherwise expr1. Dissimilar to it first parameter cannot be void.

Syntax:

NULLIF ( expr1, expr2 )

In this the first parameter can be nearer to NULL, but not as NULL. Both the constraints are compulsory for its execution.

The under query yields NULL until values, 16 are equal to each other.

Select NULLIF (16, 16) from dual;

Also, under query yields 'ABC' since both the strings are not equal.

```sql
SELECT NULLIF ('ABC', 'MOON') FROM DUAL;
```

**COALESCE**

It is basic form of NVL that gives the first non-void phrase in the parameter list. It requires minimum two parameters but there is no limit on its maximum limit.

Syntax:

COALESCE ( stmt1, stmt2, ... stmt_n )

Considering the SELECT query. The first not null data served into address domain for the employee.

```
SELECT COALESCE (address1, address2, address3) Address FROM employees;
```

The functioning of coalesce function is like to IF..ELSIF..ENDIF construct.
IF address1 =!NULL then
result := address1;
ELSIF address2 =!null THEN
result := address2;
ELSIF address3 =!null THEN
result := address3;
ELSE
result := null;
END IF;

Functions: Conditional
Two functions DECODE and CASE are used in SQL statement.

1. DECODE function:
The function is similar to conditional statement IF..THEN..ELSE.
Syntax:
```
DECODE (exp, srch, output [, search, result]... [, default])
```

DECODE checks in sequence. If equality occurs between statement and search parameter, and it yields the conforming result. If no matches occurs then null is defined. In case types mismatch then oracle within does likely inbuilt alteration to yield the results. Oracle says two null values can be same in case of decode function.

```
SELECT DECODE(NULL,NULL,'EQUAL','NOT EQUAL')
FROM DUAL;
```

If NULL expression is found, then Oracle returns output of first search as null. The No. of components are 255.

Select first_name, salary, DECODE (hire_date, sysdate,'NEW JOINEE','EMPLOYEE') FROM employees;

CASE expression
Its mechanism logically similar to DECODE but varies in format and utilization.

Syntax:
```
CASE [ expression ]
When 1_condition ... result_1
When 2_condition ... result_2
.............
When n_condition ... result_n
ELSE output
END
```

The determined number of parameters in a CASE expression are 255. Each WHEN ... THEN pair calculates as two arguments. To evade exceeding the limit, nested CASE expressions can be used so that the output_exp itself is a CASE expression.

```
Select first_name, CASE when salary < 100 THEN 'GRADE 1'
when salary > 100 AND salary < 4000 then 'GRADE 2'
ELSE 'GRADE 3'
END CASE
```

From employees;
ENAM
-----
Admin GRADE 2
Jass GRADE 3
Kumar GRADE 1

V. CONCLUSION
The Query processing of SQL functions comprises of conversion functions has done in this paper. This showed the data manipulation, formatting, general functions, conditional functioning and its transformation from inbuilt to forceful conversion. In future the work can be done on multiple row functions also.

REFERENCES

AUTHOR PROFILE

Kamalinder Kaur is working currently as Assistant Professor in Chandigarh Engineering College, Punjab, India. She has five years of teaching experience, her research interest includes Networking with specialization in Mobile Ad-hoc Network (MANET).

© 2017 by the author(s); licensee Empirical Research Press Ltd. United Kingdom. This is an open access article distributed under the terms and conditions of the Creative Commons by Attribution (CC-BY) license. (http://creativecommons.org/licenses/by/4.0/).