May 17, 2022

ISO/IEC JTC1/SC32/WG3

ANSI INCITS DM32

Database

Title: Introduction of LPAD and RPAD string padding functions

Author: Gerald Venzl

Project: SQL: 202X-CD

Status: Proposal

References:

[Foundation CD] WG3:W05-003R1 (DM32-2020-00412R1) *(ISO Committee Draft) Database Language SQL - Part 2: Foundation (SQL/Foundation),* Jim Melton (editor), November 2020.

[Foundation IWD] WG3:RKE-003 (DM32-2022-00098) *(ISO Committee Draft) Database Language SQL - Part 2: Foundation (SQL/Foundation),* Jim Melton (editor), February 2022.

[CD comments] WG3:RKE-01, Jim Melton (editor)

1 Introduction

This paper addresses the comment:

> #50 - P02-USA-130
> 1-Major Technical
> P02-06.32, <string value function>
> The SQL standard should provide RPAD and LPAD functions that extend a given string to a certain length by either adding one or more characters to the right or left of the value.
> For example (the first argument is the string to be extended, the second argument is the length of the resulting value, and the last argument is the string (which can be a single character or multiple characters) with which to extend the first argument):
> RPAD(' ', 5, '*')
> Returns (a string of length 5 – a single space followed by 4 asterisks:
> ****
>
> Solution: None provided with comment.

1.1 Rationale

For the human eye to quickly parse information, it is often required to provide visual aids. Indeed, many programming languages provide a multitude of string manipulation functions out of the box to enable software developers with the means to easily illustrate information more suitably manner to an end-user.

Understandably, the SQL standard also provides many string value functions. However, it lacks string value functions for padding a given string with additional characters to a specified length. This can be particularly useful, for example, to:

a) Provide unified output length formatting such as ensuring a column always has the same width, regardless of the value retrieved, usually padded with a white space character.

b) Illustrate hierarchies or other attributes of different data points to the human eye.

c) Provide a simple and cool way for some forms of ASCII art.

This paper proposes two string padding functions for the SQL standard:

- RPAD to pad to the right of the string.
- LPAD to pad to the left of the string.

1.2 Examples

There are many examples where string padding functionality can be useful, below are a few of these.

1.2.1 Unified output length

A common scenario is to provide the caller of an SQL statement with a unified output length of a non-fixed-width character string or other data type column. Let's assume the following example where an application would like to provide a report of all employee names and their salaries prefixed with leading '+' characters to avoid misinterpretation or accidental manipulation of the salary value. Although the salary is stored as a numeric data type, the conversion to a string data type is acceptable as the output is a printed report to the business. Using the below proposed LPAD function, this requirement can be easily accomplished:

```
SELECT first_name, last_name, LPAD(CAST(salary AS VARCHAR(20)), 7, '+') AS salary
 FROM employees
  FETCH FIRST 10 ROWS ONLY;

FIRST_NAME        LAST_NAME             SALARY
----------------- -------------------- -------
Steven            King                 ++24000
Neena             Kochhar              ++17000
Lex               De Haan              ++17000
Alexander         Hunold               +++9000
Bruce             Ernst                +++6000
David             Austin               +++4800
Valli             Pataballa            +++4800
Diana             Lorentz              +++4200
Nancy             Greenberg            ++12008
Daniel            Faviet               +++9000
```

In the above output, the salary column is first converted into a string and then padded with leading '+' characters to a maximum column value output of seven characters (accommodating salaries of up to single digit million). Regardless of what number is stored in the column itself, the width of the column will always be seven characters.

Not shown in the example above but both the total pad length and padding value can be expressions, providing further flexibility for the user as to how long and with which string to pad the pad source. The examples below show expressions being used for the total pad length.

1.2.2 Illustrate hierarchies or other attributes of different data points

This example illustrates the effect of visual aids for the human eye. A string padding function makes it easy to illustrate how big a given value actually is via simplistic ASCII formatting. Take the output of this query below:

```
SELECT name, area_sq_km / 1000 AS area
 FROM countries
  WHERE region_id = 'SA'
   ORDER BY area_sq_km / 1000 DESC
    FETCH FIRST 10 ROWS ONLY;

NAME                AREA
_____________ ____________
Brazil            8515.77
Argentina          2780.4
Peru             1285.216
Colombia           1138.91
Bolivia          1098.581
Venezuela           912.05
Chile              756.102
Paraguay           406.752
Ecuador            283.561
Guyana             214.969
```

This output represents the ten largest countries in South America by their area in 1,000 km^2.
To the human eye, this output is rather hard to parse at first and difficult to understand how big a country actually is compared to the others.

Now let's add a symbolic character for every 1,000,000 km^2, say a '*', to the output:

```
SELECT name, area_sq_km / 1000 AS area,
   LPAD('*', CAST (area_sq_km / 1000 / 1000 AS INTEGER), '*') m_sq_km
 FROM countries
  WHERE region_id = 'SA'
   ORDER BY area_sq_km / 1000 DESC
    FETCH FIRST 10 ROWS ONLY;

NAME                AREA M_SQ_KM
_____________ ____________ _________
Brazil            8515.77 ********
Argentina          2780.4 **
Peru             1285.216 *
Colombia           1138.91 *
Bolivia          1098.581 *
Venezuela           912.05
Chile              756.102
Paraguay           406.752
Ecuador            283.561
Guyana             214.969
```

While perhaps simplistic in nature, one can see how powerful such a visual aid is. It becomes immediately obvious to the human brain that Brazil is much larger than any other country. The same applies to any numeric range, whether bytes, characters typed, orders received, etc.

There are, of course, many other examples. Below is another one that illustrates the hierarchy of recursive outputs, in this case, an organizational chart (org chart). Such a chart can easily be generated via a recursive common table expression. This first statement is a recursive query that will retrieve an (incomplete) org chart. The hierarchy is indicated via a “lvl” level number. Level 0 is the top management level, while level 1, 2, etc. indicate how many levels below the top management level an employee is within the organization:

```
WITH org (employee_id, name, manager_id, lvl)
AS
(
   SELECT employee_id, first_name || ' ' || last_name AS name, manager_id, 0 AS lvl
    FROM employees
     WHERE manager_id IS NULL
   UNION ALL
   SELECT e.employee_id, e.first_name || ' ' || e.last_name AS name,
           e.manager_id, o.lvl+1 AS lvl
    FROM org o JOIN employees e ON (e.manager_id = o.employee_id)
)
SELECT name AS employee_name, lvl FROM org
 FETCH FIRST 20 ROWS ONLY;

EMPLOYEE_NAME           LVL
_____________________ _____
Steven King               0
Neena Kochhar             1
Gerald Venzl              1
Den Raphaely              1
Matthew Weiss             1
Adam Fripp                1
Payam Kaufling            1
Shanta Vollman            1
Kevin Mourgos             1
John Russell              1
Karen Partners            1
Alberto Errazuriz         1
Gerald Cambrault          1
Eleni Zlotkey             1
Michael Hartstein         1
Nancy Greenberg           2
Jennifer Whalen           2
Susan Mavris              2
Hermann Baer              2
Shelley Higgins           2
```

Below is the same query but instead of relying on a numeric output, it uses indentation to aid the human eye. For each level below the top management level, the name is indented by 3 (three) white spaces to the left:

```
WITH org (employee_id, name, manager_id, lvl)
AS
(
   SELECT employee_id, first_name || ' ' || last_name AS name, manager_id, 0 AS lvl
    FROM employees
     WHERE manager_id IS NULL
   UNION ALL
   SELECT e.employee_id, e.first_name || ' ' || e.last_name AS name,
           e.manager_id, o.lvl+1 AS lvl
    FROM org o JOIN employees e ON (e.manager_id = o.employee_id)
)
SELECT LPAD(' ', 3*lvl, ' ') || name AS employee_name FROM org
 FETCH FIRST 20 ROWS ONLY;

EMPLOYEE_NAME
_____________________________
Steven King
   Neena Kochhar
   Gerald Venzl
   Den Raphaely
   Matthew Weiss
   Adam Fripp
   Payam Kaufling
   Shanta Vollman
   Kevin Mourgos
   John Russell
   Karen Partners
   Alberto Errazuriz
   Gerald Cambrault
   Eleni Zlotkey
   Michael Hartstein
      Nancy Greenberg
      Jennifer Whalen
      Susan Mavris
      Hermann Baer
      Shelley Higgins
```

1.2.3 ASCII art via SQL

Something that the author of this proposal takes pleasure in is that padding functions also allow for some nice ASCII art (i.e. a way to generate artistic output via ASCII characters) within the SQL language. Below is a query that the author produces every year for Christmas to promote the usage of the SQL language. Incidentally, it highlights the combined usage of LPAD and RPAD as well:

```
WITH tree(lev, xmas) AS (
  SELECT 1 lev, RPAD(' ', 10, ' ')
   || '*' xmas
    FROM dual
  UNION ALL
  SELECT tree.lev+1 lev,
    RPAD(' ',10-tree.lev,' ')
     || RPAD('^',tree.lev+1,'^')
     || LPAD('^',tree.lev,'^') xmas
      FROM tree
        WHERE tree.lev < 10
)
SELECT '   Merry Christmas!'
  FROM dual
UNION ALL
SELECT xmas
  FROM tree
UNION ALL
SELECT '         | |'
  FROM dual
UNION ALL
SELECT '     ~~~/   \~~~'
  FROM dual;
```

```
   Merry Christmas!
          *
         ^^^
        ^^^^^
       ^^^^^^^
      ^^^^^^^^^
     ^^^^^^^^^^^
    ^^^^^^^^^^^^^
   ^^^^^^^^^^^^^^^
  ^^^^^^^^^^^^^^^^^
 ^^^^^^^^^^^^^^^^^^^
         | |
     ~~~/   \~~~
```

1.3 Implications of padding functions

1.3.1 LPAD and RPAD vs. just PAD

A question that may arise by the reader as to why this functionality could not be accomplished with a single function and two parameters respective to the left-side and right-side padding value. Such an implementation would not be able to determine which padding direction has priority, which is especially important for when the total pad length exceeds the padding values. That is to say, the function has no way to knowing whether the value should be padded first on the left-hand side or the right-hand side. Also, research by the author has shown that LPAD and RPAD implementations exist in at least the following database:

a. Oracle Database
b. MySQL
c. Db2
d. Snowflake
e. Postgres

1.3.2 What if the padded return string were to be longer than the total pad length

As both the pad source and the padding value can and often will be unknown to the user, situations can arise where the padded return string would be longer than the total pad length. In such cases, as the function provides a total pad length parameter, a user will, for obvious reasons, expect that this total length value will not be exceeded. The return string should be cut to accommodate the total pad length, as illustrated in the example below:

```
SELECT RPAD('abc',5,'xyz') FROM dual;

RPAD('ABC',5,'XYZ')
_____________________
abcxy
```

Priority is given to the pad source meaning that the reduction of characters has to occur in the padding value, not the overall return string, for example:

```
SQL> SELECT LPAD('abc', 5, 'xyz') FROM dual;

LPAD('ABC',5,'XYZ')
_____________________
xyabc
```

If the pad source, which can be a character value expression, exceeds the total pad length, the padding function should return a string extracted from the pad source with the total length to the total pad length. This way, a user can use the padding function on any values without having to check first whether a value needs padding or not, and can rely on that the result will never exceed the total pad length, for example:

```
SELECT RPAD('abc', 2, 'xyz') FROM dual;

RPAD('ABC',2,'XYZ')
_____________________
ab
```

1.3.3 Default value for padding value

It makes sense for the padding value to have a default value for the most common and simplest type of operations, padding a given input string up to a certain length with white spaces. A reasonable default value for the padding value hence would be a single white space character:

```
SELECT LPAD('abc', 6) FROM dual;

LPAD('ABC',6)
_____________
   abc
```

Note that the 4th example in *1.2.2 Illustrate hierarchies or other attributes of different data points* could have hence been also written without providing a white space character as padding value.

1.4 Conventions

SMALLCAPS	denote numbered editorial instructions;
New Text	New text is identified by using a red bold font;
~~Deleted Text~~	Deleted text is identified by using a blue strikethrough font;
Plain	denotes existing text to be retained;
[Note]	denotes notes to the editor.

2 Proposal

2.1 Proposed changes for [Foundation IWD]

2.1.1 Changes to 4.3.3.2 Operators that operate on character strings and return character strings

1. ADD NEW PARAGRAPH AS SHOWN HERE

<pad function> is a function that returns its first string argument padded to the total length of characters specified by the second argument with the sequence of characters specified by the third argument.

2.1.2 Changes to 5.2 <token>

2. EXPAND <RESERVED WORD> ::= WITH ADDITIONAL WORDS AS SHOWN HERE

```
| LAG | LANGUAGE | LARGE | LAST_VALUE | LATERAL | LEAD | LEADING | LEAST | LEFT
| LIKE | LIKE_REGEX | LISTAGG | LN | LOCAL | LOCALTIME | LOCALTIMESTAMP | LOG
| LOG10 | LOWER | LPAD

| RANGE | RANK | READS | REAL | RECURSIVE | REF | REFERENCES | REFERENCING
| REGR_AVGX | REGR_AVGY | REGR_COUNT | REGR_INTERCEPT | REGR_R2 | REGR_SLOPE
| REGR_SXX | REGR_SXY | REGR_SYY | RELEASE | RESULT | RETURN | RETURNS
| REVOKE | RIGHT | ROLLBACK | ROLLUP | ROW | ROW_NUMBER | ROWS | RPAD | RUNNING
```

2.1.3 Changes to 6.33 <string value function>

1. EXPAND <CHARACTER VALUE FUNCTION> ::= BNF WITH ADDITIONAL OPTION AS SHOWN HERE:

```
<character value function> ::=
    <character substring function>
  | <regular expression substring function>
  | <regex substring function>
  | <fold>
  | <transcoding>
  | <character transliteration>
  | <regex transliteration>
  | <trim function>
  | <pad function>
  | <character overlay function>
  | <normalize function>
  | <specific type method>
  | <classifier function>
```

2. ADD THE FOLLOWING BNFS AFTER THE <TRIM CHARACTER> BNF:

<pad function> ::=
{ LPAD | RPAD } <left paren> <pad operands> <right paren>

<pad operands> ::=
<pad source> <comma> <total pad length> [<comma> <padding character value expression>]

<pad source> ::=
<character value expression>

<total pad length> ::=
<numeric value expression>

<padding character value expression> ::=
<character value expression>

3. CHANGE SR 2 AS SHOWN HERE:

2) The declared type of <character value function> is the declared type of the immediately contained <character substring function>, <regular expression substring function>, <regex substring function>, <fold>, <transcoding>, <character transliteration>, <regex transliteration>, <trim function>, **<pad function>,** <character overlay function>, <normalize function>, <specific type method>, or <classifier function>.

4. ADD NEW SYNTAX RULE AS SHOWN HERE:

N) If <pad function> is specified, then:
a) If <padding character value expression> is not specified, then <space> is implicit.
b) The declared type of <total pad length> shall be an exact numeric type with scale 0 (zero).
b) Case:
i) If the declared type of <character value expression> simply contained in <pad source> is fixed-length character string or variable-length character string, then the declared type of the <pad function> is variable-length character string with implementation-defined maximum length *LR*.
ii) Otherwise, the declared type of the <pad function> is a character large object type with implementation-defined maximum length *LR*.
c) <padding character value expression> and <pad source> shall be comparable.
d) The character set and collation of the <pad function> are those of the <pad source>.

5. CHANGE SR 13, A) I) 3)

If <trim character> is not specified, then ' ' **<space>** is implicit.

Note to the editor: bug fix on the fly: do not use ' ' but instead the <space> SQL terminal character

6. MODIFY GR 2 AS SHOWN HERE:

The result of <character value function> is the result of the immediately contained <character substring function>, <regular expression substring function>, <regex substring function>, <fold>, <transcoding>, <character transliteration>, <regex transliteration>, <trim function>, **<pad function>,** <character overlay function>, <normalize function>, <specific type method>, or <classifier function>.

7. CHANGE GR 3, D) AS SHOWN HERE:

If at least one of *C*, *S*, and *L* is the null value, then the result of the <character substring function> is the null value **and no further General Rules of this Subclause are applied**.

Note to the editor: bug fix on the fly for <character substring function>

8. ADD NEW GENERAL RULE AS SHOWN HERE:

N) If <pad function> is specified, then:

- **a) Let *PS* be the value of <pad source>. If *PS* is the null value, then the result of <pad function> is the null value and no further General Rules of this Subclause are applied.**
- **b) Let *TPL* be the value of <total pad length>. If *TPL* is the null value, then the result of <pad function> is the null value and no further General Rules of this Subclause are applied.**
- **c) If TPL is less than 1 (one), then the result of <pad function> is the zero-length character string and no further General Rules of this Subclause are applied.**
- **d) Let *PV* be the value of <pad character value expression>. If *PV* is the null value, then the result of <pad function> is the null value and no further General Rules of this Subclause are applied.**
- **e) Let *LPS* be the length of *PS*.**
- **f) Let *LPV* be the maximum of 1 (one) and the length of *PV*.**
- **g) Let *MRL* be the minimum of *TPL* and *LR*.**
- **h) Let *MPL* be *MRL – LPS*.**
- **i) If *MPL* is 0 (zero), then the result of the <pad function> is *PS* and no further General Rules of this Subclause are applied.**
- **j) If *MPL* is negative, then the result of the <pad function> is the first *MRL* characters of *PS* and no further General Rules of this Subclause are applied.**
- **k) Let *PI* be the greatest integer less than or equal to *MPL* / *LPV*.**
- **l) Let *PR* be the value of MOD (*MPL*, *LPV*).**
- **m) Let D_0 be the zero-length character string. For *i*, 1 (one) <= *i* <= *PI*, let D_i be D_{i-1} || *PV*.**
  Note to the reader: D_{PI} is a concatenation of PV with itself as many times as it fits fully into the result.
- **n) If *PR* is greater than 0 (zero), then let *PVR* be the first *PR* characters of *PV*; otherwise, let *PVR* be the zero-length character string.**
  Note to the reader: PVR is the substring of PV that fits into the remainder of the result.
- **o) The result of the <pad function> is**
  Case:
  - **i) If RPAD is specified, then *PS* || D_{PI} || *PVR*.**
  - **ii) Otherwise (LPAD is specified), D_{PI} || *PVR* || *PS*.**

9. ADD NEW CONFORMANCE RULE AS SHOWN HERE:

N) Without Feature FNNN, "String padding functions", conforming SQL language shall not contain a <pad function>.

2.1.4 Changes to Annex B, 54) Subclause 6.33, "<string value function>",

1. MODIFY AS SHOWN HERE FOLLOWING AS SHOWN HERE:

a) The maximum length of <character transliteration>, **<pad function>**, <transcoding>, <normalize function> without <normalize function result length> specified, or <specific type method>, is implementation-defined.

2.1.5 Changes to Annex D, Table 45 – Feature taxonomy for optional features

1. ADD NEW ROW:

N | FNNN | String padding functions

2.1.6 Changes to Annex F, 1)

1. ADD LPAD AND RPAD TO <RESERVED WORDS> AS SHOWN BELOW:

— ABSENT

— ANY_VALUE

— GREATEST

— JSON

— JSON_SCALAR

— JSON_SERIALIZE

— LEAST

— LPAD

— PIPE

— RPAD

2.2 Proposed changes for [Foundation CD]

2.2.1 Apply the same changes as in section 2.1

3 Checklist

Interactions with other concurrent proposals identified and editorial assistance given	N/A
Concepts	Yes
Access Rules	N/A
Conformance Rules	Yes
Lists of SQL- statements by category	N/A
Table of identifiers used by diagnostics statements	N/A
Collation coercibility for character strings	N/A
Closing Possible Problems	Yes
Any new Possible Problems clearly identified	N/A
Reserved and non- reserved keywords	Yes
SQLSTATE tables and Ada package	N/A
Information and Definition Schemas	N/A
Implementation- defined and –dependent Annexes	Yes
Incompatibilities Annex	Yes
Embedded SQL and host language implications	N/A
Dynamic SQL issues: including descriptor areas	N/A
PSM impact	None
Schemata impact	None
XML impact	None
PGQ impact	None

- End of paper -