SQL Server Mathematical functions (SQRT, PI, SQUARE, ROUND, CEILING & FLOOR)

Sam007

Mathematical functions are present in SQL server which can be used to perform mathematical calculations. Some commonly used mathematical functions are given below:

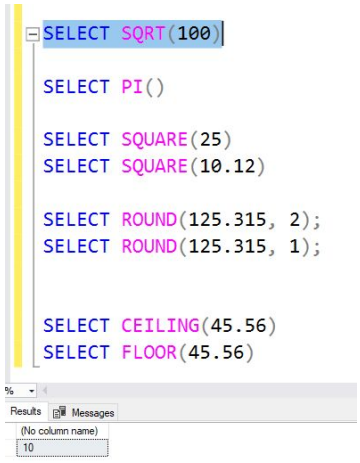
1. SQRT():

SQRT() function is the most commonly used function. It takes any numeric values and returns the square root value of that number.

Syntax:

```
SELECT SQRT(..value..)
```

Example:



2. PI(): There are calculations which require use of pi. Using pi() function, value of PI can be used anywhere in the query.

Syntax:

```
SELECT PI()
```

Example:

```
SELECT SQRT(100)

SELECT PI()

SELECT SQUARE(25)
SELECT SQUARE(10.12)

SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);


SELECT CEILING(45.56)
SELECT FLOOR(45.56)
```

Results Messages

(No column name)

3.14159265358979

3. SQUARE(): SQUARE() function is used to find the square of any number.

Syntax:

```
SELECT SQUARE(..value..)
```

Example:

```
SELECT SQUARE(25)
SELECT SQUARE(10.12)

SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);


SELECT CEILING(45.56)
SELECT FLOOR(45.56)
```

Results Messages

(No column name)

625

(No column name)

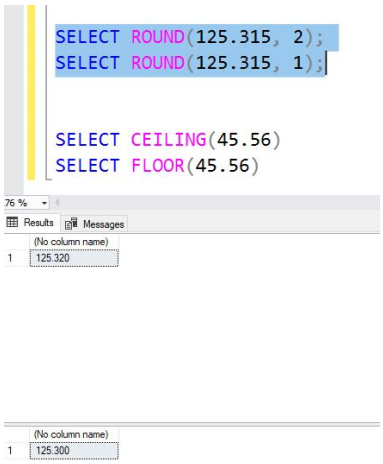
102.4144

4. ROUND(): ROUND() function is used to round a value to the nearest specified decimal place.

Syntax:

```
SELECT ROUND(..value.., number_of_decimal_places)
```

Example:



5. CEILING() and FLOOR()

CEILING(): CEILING() function is used to find the next highest value (integer).

Syntax:

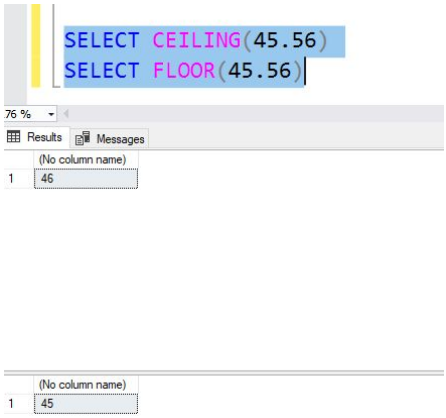
```
SELECT CEILING(..value..)
```

FLOOR(): FLOOR() function returns the next lowest value (integer).

Syntax:

```
SELECT FLOOR(..value..)
```

Example:



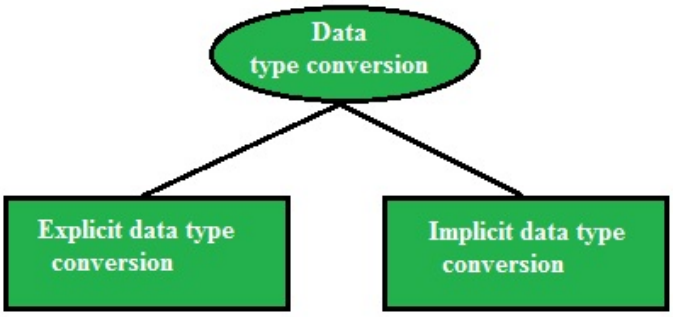
Last Updated : 21 Mar, 2018

16

Similar Reads

SQL | Conversion Function

MrinalVerma



In some cases, the Server uses data of one type where it expects data of a different data type. This can happen when the Server can automatically convert the data to the expected data type. This data type conversion can be done implicitly by the Server, or explicitly by the user.

Implicit Data-Type Conversion :

In this type of conversion the data is converted from one type to another implicitly (by itself/automatically).

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
DATE	VARCHAR2
NUMBER	VARCHAR2

1. **QUERY:**

```
SELECT employee_id,first_name,salary
FROM employees
WHERE salary > 15000;
```

1. **OUTPUT :**

Employee_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	lex	17000

1. **QUERY:**

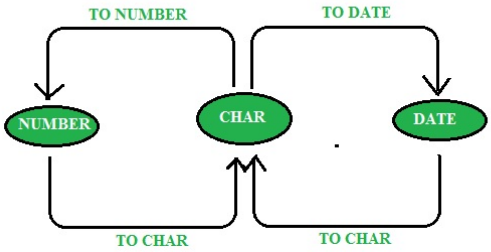
```
SELECT employee_id,first_name,salary
FROM employees
WHERE salary > '15000';
```

1. **OUTPUT :**

Employee_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	lex	17000

1. Here we see the output of both queries came out to be same,inspite of 2nd query using **'15000'** as text, it is automatically converted into **int** data type.

Explicit Data-Type Conversion :



Explicit Data Type Converson

TO_CHAR Function :

TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

SYNTAX :

```
TO_CHAR(number1, [format], [nls_parameter])
```

Using the TO_CHAR Function with Dates :

SYNTAX :

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed in single quotation marks and is case sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

EXAMPLE :

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees
WHERE last_name = 'Higgins';
```

OUTPUT :

EMPLOYEE_ID	MONTH_HIRED
205	06/94

Elements of the Date Format Model :

YYYY	Full year in Numbers
YEAR	Year spelled out
MM	Two digit value for month
MONTH	Full name of the month
MON	Three Letter abbreviation of the month
DY	Three letter abbreviation of the day of the week
DAY	Full Name of the Day
DD	Numeric day of the month

Elements of the Date Format Model :

Date Format Elements – Time Formats :

Use the formats listed in the following tables to display time information and literals and to change numerals to spelled numbers.

ELEMENT	DESCRIPTION
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day,or hour (1-12),or hour (0-23)
MI	Minute 0-59
SS	Second 0-59
SSSSS	Second past Mid Night 0-86399

Other Formats :

ELEMENT	DESCRIPTION
/ . ,	Punctuation is reproduced in the result
“of the”	Quoted string is reproduced in the result

Specifying Suffixes to Influence Number Display :

ELEMENT	DESCRIPTION
TH	Ordinal Number (for example DDTH for 4TH
SP	Spelled out number (for example DDSP for FOUR
SPTH or THSP	spelled out ordinal numbers (for example DDSPTH for FOURTH

EXAMPLE :

```
SELECT last_name,
TO_CHAR(hire_date, 'fmDD Month YYYY')
AS HIREDATE
FROM employees;
```

OUTPUT :

LASTNAME	HIIREDATE
Austin	25 January 2005
Shubham	20 June 2004
Nishant	15 January 1999
Ankit	15 July 1995
Vanshika	5 August 2004
Kusum	10 June 1994
Faviet	11 March 2005
King	9 April 1996

Using the TO_CHAR Function with Numbers :

SYNTAX :

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements you can use with the TO_CHAR function to display a number value as a character :

9	Represent a number
0	Forces a zero to be displayed

$	places a floating dollar sign
L	Uses the floating local currency symbol
.	Print a decimal point
,	Prints a Thousand indicator

EXAMPLE :

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name = 'Ernst';
```

OUTPUT :

SALARY
$5000

Using the TO_NUMBER and TO_DATE Functions :

Convert a character string to a number format using the **TO_NUMBER** function :

```
TO_NUMBER(char[, 'format_model'])
```

Convert a character string to a date format using the **TO_DATE** function:

```
TO_DATE(char[, 'format_model'])
```

These functions have an **fx** modifier. This modifier specifies the exact matching for the character argument and date format model of a **TO_DATE** function.

EXAMPLE :

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

OUTPUT :

LASTNAME	HIREDATE
Kumar	24-MAY-99

Last Updated : 28 Mar, 2023

16

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Configure SQL Jobs in SQL Server using T-SQL
3. MS SQL Server - Type Conversion
4. SQL | Procedures in PL/SQL
5. SQL | Difference between functions and stored procedures in PL/SQL
6. SQL SERVER – Input and Output Parameter For Dynamic SQL
7. Difference between SQL and T-SQL
8. SQL Server | Convert tables in T-SQL into XML
9. SQL SERVER | Bulk insert data from csv file using T-SQL command
10. SQL - SELECT from Multiple Tables with MS SQL Server

Previous Next

Article Contributed By :

MrinalVerma
MrinalVerma

Vote for difficulty

Current difficulty : Easy

SQL general functions | NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL

shubham_rana_77

In this article, we'll be discussing some powerful SQL general functions, which are – NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL.

These functions work with any data type and pertain to the use of null values in the expression list. These are all **single row function** i.e. provide one result per row.

- **NVL(expr1, expr2)** : In SQL, NVL() converts a null value to an actual value. Data types that can be used are date, character and number. Data type must match with each other i.e. expr1 and expr2 must of same data type.
  Syntax –

```
NVL (expr1, expr2)
```

  expr1 is the source value or expression that may contain a null.
  expr2 is the target value for converting the null.

  Example –

```
SELECT  salary, NVL(commission_pct, 0),
    (salary*12) + (salary*12*NVL(commission_pct, 0))
      annual_salary FROM employees;
```

  Output :

SALARY	COMMISSION_PCT	NVL(COMMISSION_PCT,0)	ANNUAL_SALARY
8400	.2	.2	120960
7000	.15	.15	96600
6200	.1	.1	81840
3200		0	38400
3100		0	37200
2500		0	30000

- **NVL2(expr1, expr2, expr3)** : The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned i.e. If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.
  Syntax –

```
NVL2 (expr1, expr2, expr3)
```

  expr1 is the source value or expression that may contain null
  expr2 is the value returned if expr1 is not null
  expr3 is the value returned if expr1 is null

  Example –

```
SELECT last_name, salary, commission_pct,
 NVL2(commission_pct, 'SAL+COMM', 'SAL')
 income FROM employees;
```

  Output :

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Livingston	8400	.2	SAL+COMM
Grant	7000	.15	SAL+COMM
Johnson	6200	.1	SAL+COMM
Taylor	3200		SAL
Fleaur	3100		SAL
Sullivan	2500		SAL

- **DECODE()** : Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement.
  The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic

used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.
If the default value is omitted, a null value is returned where a search value does not match any of the result values.
Syntax –

```
DECODE(col|expression, search1, result1
 [, search2, result2,...,][, default])
```

Example –

```
SELECT last_name, job_id, salary,
   DECODE(job_id, 'IT_PROG', 1.10*salary,
    'ST_CLERK', 1.15*salary,
    'SA_REP', 1.20*salary,salary)
     REVISED_SALARY FROM employees;
```

Output :

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
Lorentz	IT_PROG	4200	4620
Sarchand	SH_CLERK	4200	4200
Whalen	AD_ASST	4400	4400
Austin	IT_PROG	4800	5280
Pataballa	IT_PROG	4800	5280
Mourgos	ST_MAN	5800	5800
Ernst	IT_PROG	6000	6600
Fay	MK_REP	6000	6000
Kumar	SA_REP	6100	7320
Banda	SA_REP	6200	7440

- **COALESCE()** : The COALESCE() function examines the first expression, if the first expression is not null, it returns that expression; Otherwise, it does a COALESCE of the remaining expressions.
  The advantage of the COALESCE() function over the NVL() function is that the COALESCE function can take multiple alternate values. In simple words COALESCE() function returns the first non-null expression in the list.
  Syntax –

```
COALESCE (expr_1, expr_2, ... expr_n)
```

Examples –

```
SELECT last_name,
    COALESCE(commission_pct, salary, 10) comm
    FROM employees ORDER BY commission_pct;
```

Output :

LAST_NAME	COMM
Livingston	.2
Grant	.15
Johnson	.1
Taylor	3200
Fleaur	3100
Sullivan	2500
Geoni	2800
Sarchand	4200
Bull	4100
Dellinger	3400
Cabrio	3000

- **NULLIF()** : The NULLIF function compares two expressions. If they are equal, the function returns null. If they are not equal, the function returns the first expression. You cannot specify the literal NULL for first expression.
  Syntax –

```
NULLIF (expr_1, expr_2)
```

Examples –

```
SELECT LENGTH(first_name) "expr1",
   LENGTH(last_name) "expr2",
   NULLIF(LENGTH(first_name),LENGTH(last_name))
    result FROM employees;
```

Output :

LENGTH(FIRST_NAME)	LENGTH(LAST_NAME)	RESULT
8	5	8
6	5	6
5	5	
7	9	7
7	7	
6	6	
3	3	
5	6	5
4	6	4
6	3	6
4	5	4

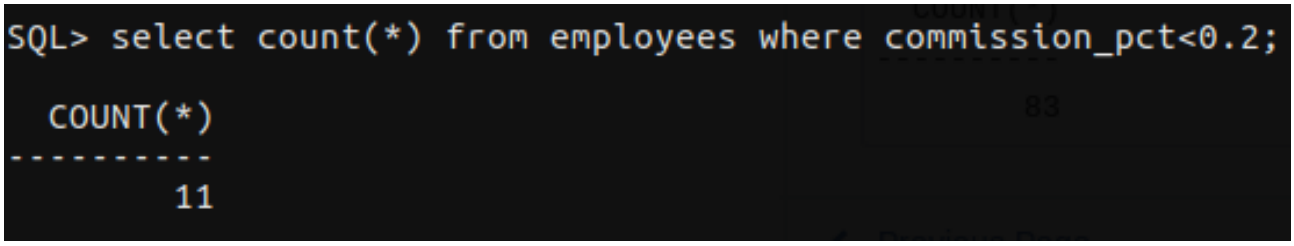
- **LNNVL()** : LNNVL evaluate a condition when one or both operands of the condition may be null. The function can be used only in the WHERE clause of a query. It takes as an argument a condition and returns TRUE if the condition is FALSE or UNKNOWN and FALSE if the condition is TRUE.
  Syntax –

```
LNNVL( condition(s) )
```

Examples –

```
SELECT COUNT(*) FROM employees
      WHERE commission_pct < .2;
```

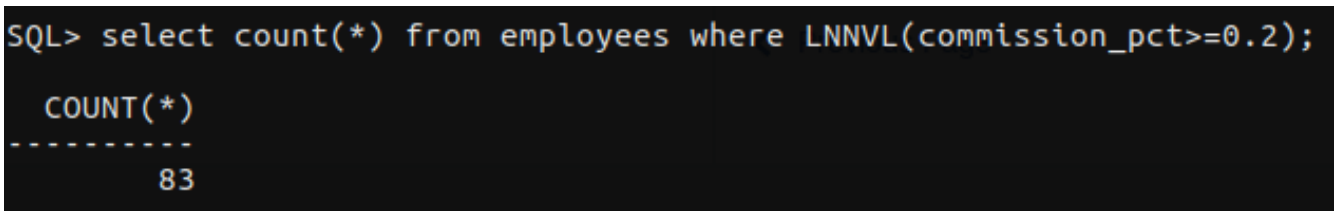
Output :



Now the above examples does not considered those employees who have no commission at all.
To include them as well we use LNNVL()

```
SELECT COUNT(*) FROM employees
  WHERE LNNVL(commission_pct >= .2);
```

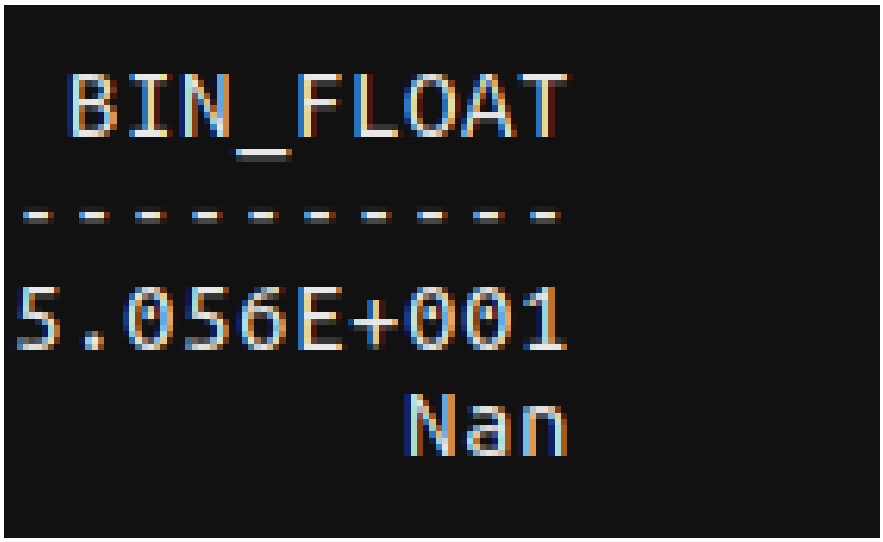
Output :



- **NANVL()** : The NANVL function is useful only for floating-point numbers of type BINARY_FLOAT or BINARY_DOUBLE. It instructs the Database to return an alternative value n2 if the input value n1 is NaN (not a number). If n1 is not NaN, then database returns n1. This function is useful for mapping NaN values to NULL.
  Syntax –

```
NANVL( n1 , n2 )
```

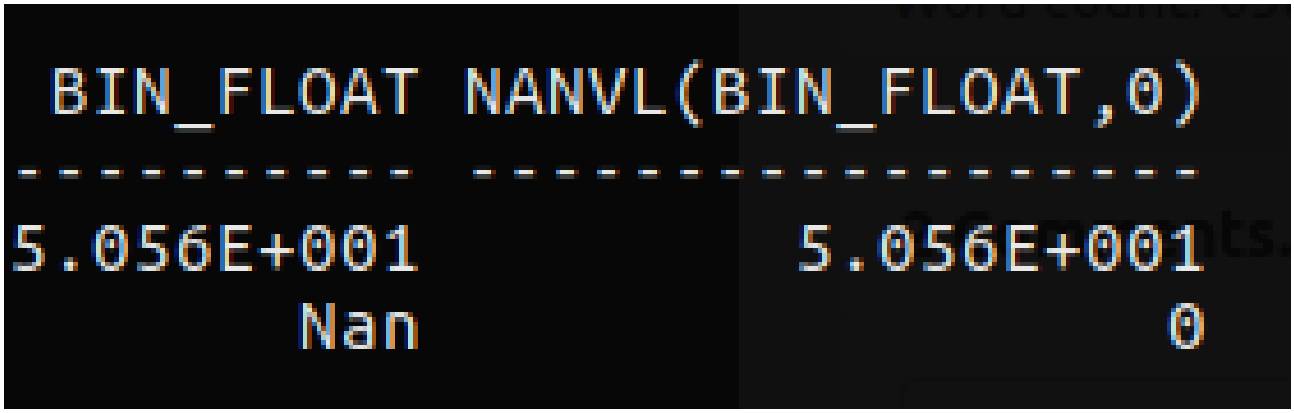
Consider the following table named nanvl_demo :



Example –

```
SELECT bin_float, NANVL(bin_float,0)
  FROM nanvl_demo;
```

Output :



Reference: Introduction to Oracle9i SQL(Volume-1 Book)

Last Updated : 18 Dec, 2019

11

Similar Reads

1. NULLIF() Function in SQL Server
2. Use of COALESCE() function in SQL Server
3. MySQL | NULLIF( ) Function

SQL | Conditional Expressions

Following are Conditional Expressions in SQL

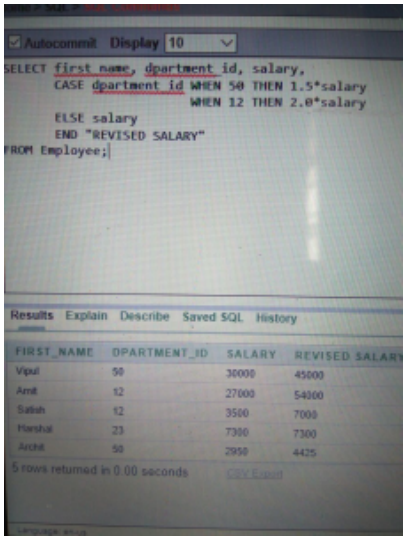
1. **The CASE Expression**: Let you use IF-THEN-ELSE statements without having to invoke procedures.
   In a simple CASE expression, the SQL searches for the first WHEN.....THEN pair for which expr is equal to comparison_expr and returns return_expr. If above condition is not satisfied, an ELSE clause exists, the SQL returns else_expr. Otherwise, returns NULL.
   We cannot specify literal null for the return_expr and the else_expr. All of the expressions(expr, comparison_expr, return_expr) must be of the same data type.
   Syntax:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
         [WHEN comparison_expr2 THEN return_expr2
          .
          .
          .
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END
```

   Example:

```
Input :
SELECT first_name, department_id, salary,
       CASE department_id WHEN 50 THEN 1.5*salary
                          WHEN 12 THEN 2.0*salary
       ELSE salary
       END "REVISED SALARY"
FROM Employee;
```

Output :



Explanation: In above SQL statements, the value of department_id is decoded. If it is 50 then salary is made 1.5 times, if it is 12 then salary is made 2 times, else there is no change in salary.

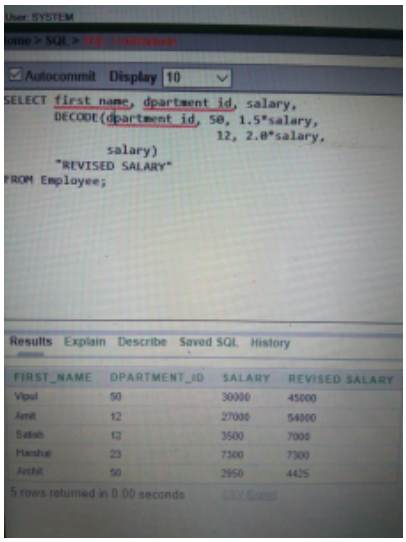
2. **The DECODE Function :** Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement.
   The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.
   If the default value is omitted, a null value is returned where a search value does not match any of the result values.
   Syntax:

```
DECODE(col/expression, search1, result1
                       [, search2, result2,........,]
                       [, default])
```

```
Input :
SELECT first_name, department_id, salary,
       DECODE(department_id, 50, 1.5*salary,
                             12, 2.0*salary,
              salary)
```

```
        "REVISED SALARY"
FROM Employee;
```

Output :



Explanation: In above SQL statements, the value of department_id is tested. If it is 50 then salary is made 1.5 times, if it is 12 then salary is made 2 times, else there is no change in salary.

3. **COALESCE :** Returns the first non-null argument. Null is returned only if all arguments are null. It is often used to substitute a default value for null values when data is retrieved for display.
   NOTE: Same as CASE expressions, COALESCE also will not evaluate the arguments to the right of the first non-null argument found.
   Syntax:

```
COALESCE(value [, ......] )
```

```
Input:
SELECT COALESCE(last_name, '- NA -')
from Employee;
```

Output:

Explanation: "- NA -" will be displayed in place where last name is null else respective last names will be shown.

4. **GREATEST:** Returns the largest value from a list of any number of expressions. Comparison is case sensitive. If datatypes of all the expressions in the list are not same, rest all expressions are converted to the datatype of the first expression for comparison and if this conversion is not possible, SQL will throw an error.
NOTE: Returns null if any expression in the list is null.
Syntax:

```
GREATEST(expr1, expr2 [, .....] )
```

- **Input:**
```
SELECT GREATEST('XYZ', 'xyz')
from dual;
```

Output:
```
GREATEST('XYZ', 'xyz')
xyz
```

Explanation: ASCII value of small alphabets is greater.

- **Input:**
```
SELECT GREATEST('XYZ', null, 'xyz')
from dual;
```

Output:
```
GREATEST('XYZ', null, 'xyz')
-
```

Explanation: Since null is present hence, null will be shown as output (as mentioned to note in description above).

5. **IFNULL:** If expr1 is not NULL, returns expr1; otherwise it returns expr2. Returns a numeric or string value, depending on the context in which it is used.
   Syntax:

```
IFNULL(expr1, expr2)
```

   - **Input:**
     ```
     SELECT IFNULL(1,0)
     FROM dual;
     ```

     Output:
     ```
     -
     1
     ```

     Explanation : Since, no expression is null.

   - **Input:**
     ```
     SELECT IFNULL(NULL,10)
     FROM dual;
     ```

     Output:
     ```
     --
     10
     ```

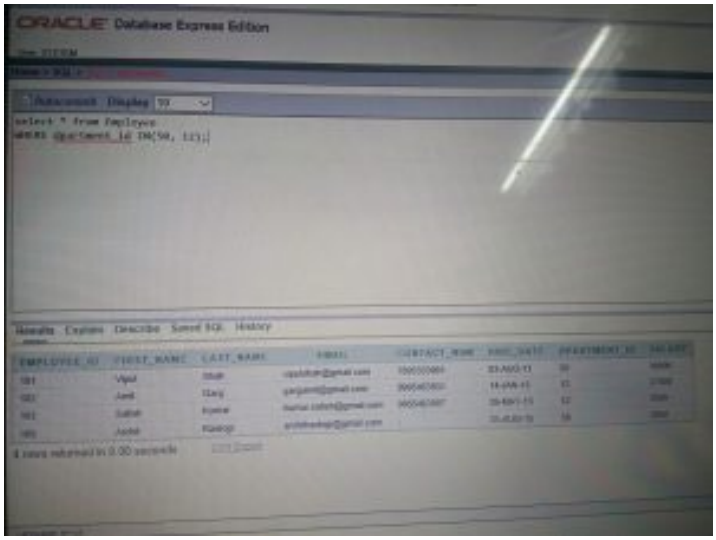
     Explanation: Since, expr1 is null hence, expr2 is shown.

6. **IN:** Checks whether a value is present within a set of values and can be used with WHERE, CHECK and creation of views.
   NOTE: Same as CASE and COALESCE expressions, IN also will not evaluate the arguments to the right of the first non-null argument found.
   Syntax:

```
WHERE column IN (x1, x2, x3 [,......] )
```

```
Input:
SELECT * from Employee
WHERE department_id IN(50, 12);
```

Output:



Explanation: All data of Employees is shown with department ID 50 or 12.

7. **LEAST:** Returns the smallest value from a list of any number of expressions. Comparison is case sensitive. If datatypes of all the expressions in the list are not same, rest all expressions are converted to the datatype of the first expression for comparison and if this conversion is not possible, SQL will throw an error.
   NOTE: Returns null if any expression in the list is null.
   Syntax:

```
LEAST(expr1, expr2 [, ......])
```

- strong>Input:
  SELECT LEAST('XYZ', 'xyz')
  from dual;

  Output:
  LEAST('XYZ', 'xyz')
  XYZ

  Explanation: ASCII value of capital alphabets is smaller.

- **Input:**
  SELECT LEAST('XYZ', null, 'xyz')
  from dual;

  Output:
  LEAST('XYZ', null, 'xyz')
  -

Explanation: Since null is present hence, null will be shown as output (as mentioned to note in description above).

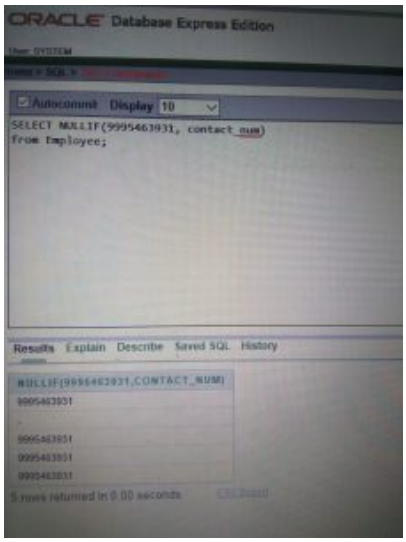
8. **NULLIF:** Returns a null value if value1=value2, otherwise it returns value1.
   Syntax:

```
NULLIF(value1, value2)
```

   Example:

```
Input:
SELECT NULLIF(9995463931, contact_num)
from Employee;
```

   Output:



Explanation: NULL is displayed for the Employee whose number is matched with the given number. For rest of the Employees value1 is returned.

This article is contributed by **akanshgupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 29 Apr, 2022

11

Similar Reads

1. SQL SERVER | Conditional Statements

SQL | Character Functions with Examples

Character functions accept character inputs and can return either characters or number values as output. SQL provides a number of different character datatypes which includes – CHAR, VARCHAR, VARCHAR2, LONG, RAW, and LONG RAW. The various datatypes are categorized into three different datatypes :

1. **VARCHAR2** – A variable-length character datatype whose data is converted by the RDBMS.
2. **CHAR** – The fixed-length datatype.
3. **RAW** – A variable-length datatype whose data is not converted by the RDBMS, but left in "raw" form.

Note : When a character function returns a character value, that value is always of type VARCHAR2 (variable length), with the following two exceptions: UPPER and LOWER. These functions convert to upper and to lower case, respectively, and return the CHAR values (fixed length) if the strings they are called on to convert are **fixed-length** CHAR arguments.

Character Functions

SQL provides a rich set of character functions that allow you to get information about strings and modify the contents of those strings in multiple ways. Character functions are of the following two types:

1. Case-Manipulative Functions (LOWER, UPPER and INITCAP)
2. Character-Manipulative Functions (CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE)

Case-Manipulative Functions

1. **LOWER :** This function converts alpha character values to lowercase. LOWER will actually return a fixed-length string if the incoming string is fixed-length. LOWER will not change any characters in the string that are not letters, since case is irrelevant for numbers and special characters, such as the dollar sign ($) or modulus (%).
   Syntax:

```
LOWER(SQL course)


Input1: SELECT LOWER('GEEKSFORGEEKS') FROM DUAL;
Output1: geeksforgeeks

Input2: SELECT LOWER('DATABASE@456') FROM DUAL;
Output2: database@456
```

2. **UPPER :** This function converts alpha character values to uppercase. Also UPPER function too, will actually return a fixed-length string if the incoming string is fixed-length. UPPER will not change any characters in the string that are not letters, since case is irrelevant for numbers and special characters, such as the dollar sign ($) or modulus (%).
   Syntax:

```
UPPER(SQL course)


Input1: SELECT UPPER('geeksforgeeks') FROM DUAL;
Output1: GEEKSFORGEEKS

Input2: SELECT UPPER('dbms$508%7') FROM DUAL;
Output2: DBMS$508%7
```

3. **INITCAP :** This function converts alpha character values to uppercase for the first letter of each word and all others in lowercase. The words in the string is must be separated by either # or _ or space.
   Syntax:

```
INITCAP(SQL course)


Input1: SELECT INITCAP('geeksforgeeks is a computer science portal for geeks')
FROM DUAL;
Output1: Geeksforgeeks Is A Computer Science Portal For Geeks

Input2: SELECT INITCAP('PRACTICE_CODING_FOR_EFFICIENCY') FROM DUAL;
Output2: Practice_Coding_For_Efficiency
```

Character-Manipulative Functions

1. **CONCAT :** This function always appends (concatenates) string2 to the end of string1. If either of the string is NULL, CONCAT function returns the non-NULL argument. If both

strings are NULL, CONCAT returns NULL.
Syntax:

```
CONCAT('String1', 'String2')
```

```
Input1: SELECT CONCAT('computer' ,'science') FROM DUAL;
Output1: computerscience

Input2: SELECT CONCAT( NULL ,'Android') FROM DUAL;
Output2: Android

Input3: SELECT CONCAT( NULL ,NULL ) FROM DUAL;
Output3: -
```

2. **LENGTH :** This function returns the length of the input string. If the input string is NULL, then LENGTH function returns NULL and not Zero. Also, if the input string contains extra spaces at the start, or in between or at the end of the string, then the LENGTH function includes the extra spaces too and returns the complete length of the string.
Syntax:

```
LENGTH(Column|Expression)
```

```
Input1: SELECT LENGTH('Learning Is Fun') FROM DUAL;
Output1: 15

Input2: SELECT LENGTH('   Write an Interview  Experience ') FROM DUAL;
Output2: 34

Input3: SELECT LENGTH('') FROM DUAL; or SELECT LENGTH( NULL ) FROM DUAL;
Output3: -
```

3. **SUBSTR :** This function returns a portion of a string from a given start point to an end point. If a substring length is not given, then SUBSTR returns all the characters till the end of string (from the starting position specified).
Syntax:

```
SUBSTR('String',start-index,length_of_extracted_string)
```

```
Input1: SELECT SUBSTR('Database Management System', 9) FROM DUAL;
Output1: Management System

Input2: SELECT SUBSTR('Database Management System', 9, 7) FROM DUAL;
Output2: Manage
```

4. **INSTR :** This function returns numeric position of a character or a string in a given string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of string.

Also, if the starting position is not given, then it starts search from index 1, by default. If after searching in the string, no match is found then, INSTR function returns 0.

```
Syntax: INSTR(Column|Expression, 'String', [,m], [n])


Input: SELECT INSTR('Google apps are great applications','app',1,2) FROM DUAL;
Output: 23
```

5. **LPAD and RPAD :** These functions return the strings padded to the left or right (as per the use) ; hence the “L” in “LPAD” and the “R” in “RPAD” ; to a specified length, and with a specified pad string. If the pad string is not specified, then the given string is padded on the left or right (as per the use) with spaces.
   Syntax:

```
LPAD(Column|Expression, n, 'String')
Syntax: RPAD(Column|Expression, n, 'String')


LPAD Input1: SELECT LPAD('100',5,'*') FROM DUAL;
LPAD Output1: **100

LPAD Input2: SELECT LPAD('hello', 21, 'geek') FROM DUAL;
LPAD Output2: geekgeekgeekgeekhello

RPAD Input1: SELECT RPAD('5000',7,'*') FROM DUAL;
RPAD Output1: 5000***

RPAD Input1: SELECT RPAD('earn', 19, 'money') FROM DUAL;
RPAD Output1: earnmoneymoneymoney
```

6. **TRIM :** This function trims the string input from the start or end (or both). If no string or char is specified to be trimmed from the string and there exists some extra space at start or end of the string, then those extra spaces are trimmed off.

```
Syntax:

 TRIM(Leading|Trailing|Both, trim_character FROM trim_source)


 Input1: SELECT TRIM('G' FROM 'GEEKS') FROM DUAL;
 Output1: EEKS

 Input2: SELECT TRIM('        geeksforgeeks   ') FROM DUAL;
 Output2:geeksforgeeks
```

7. **REPLACE :** This function searches for a character string and, if found, replaces it with a given replacement string at all the occurrences of the string. REPLACE is useful for searching patterns of characters and then changing all instances of that pattern in a single

function call.
If a replacement string is not given, then REPLACE function removes all the occurrences of that character string in the input string. If neither a match string nor a replacement string is specified, then REPLACE returns NULL.
Syntax:

```
REPLACE(Text, search_string, replacement_string)
```

```
Input1: SELECT REPLACE('DATA MANAGEMENT', 'DATA','DATABASE') FROM DUAL;
Output1: DATABASE MANAGEMENT

Input2: SELECT REPLACE('abcdeabcccabdddeeabcc', 'abc') FROM DUAL;
Output2: deccabdddeec
```

This article is contributed by **Anshika Goyal.** If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 21 Mar, 2018

20

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)
2. SQL | Difference between functions and stored procedures in PL/SQL
3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
4. Configure SQL Jobs in SQL Server using T-SQL
5. SQL | Date functions
6. SQL | NULL functions
7. SQL general functions | NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL
8. SQL Server Mathematical functions (SQRT, PI, SQUARE, ROUND, CEILING & FLOOR)
9. SQL | Numeric Functions

SQL | Date functions

In SQL, dates are complicated for newbies, since while working with a database, the format of the data in the table must be matched with the input data to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

For storing a date or a date and time value in a database,**MySQL** offers the following data types:

DATE	format YYYY-MM-DD
DATETIME	format: YYYY-MM-DD HH:MI: SS
TIMESTAMP	format: YYYY-MM-DD HH:MI: SS
YEAR	format YYYY or YY

Now, come to some popular functions in SQL date functions.

NOW()

Returns the current date and time.

Query:

```
SELECT NOW();
```

Output:

Number of Records: 1

NOW()
2023-04-04 07:29:38

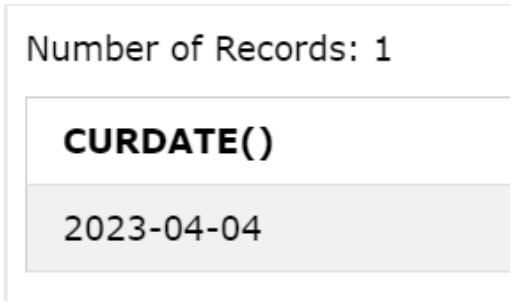
CURDATE()

Returns the current date.

Query:

```
SELECT CURDATE();
```

Output:



Number of Records: 1

CURDATE()
2023-04-04

CURTIME()

Returns the current time.

Query:

```
SELECT CURTIME();
```

Output:

Number of Records: 1

CURTIME()
07:32:24

DATE()

Extracts the date part of a date or date/time expression. Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, DATE(BirthTime)
AS BirthDate FROM Test;
```

Output:

Name	BirthDate
Pratik	1996-09-26

EXTRACT()

Returns a single part of a date/time.

Syntax

EXTRACT(unit FROM date);

Several units can be considered but only some are used such as **MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.** And 'date' is a valid date expression. Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, Extract(DAY FROM
BirthTime) AS BirthDay FROM Test;
```

Output:

Name	Birthday
Pratik	26

Query:

```
SELECT Name, Extract(YEAR FROM BirthTime)
AS BirthYear FROM Test;
```

Output:

Name	BirthYear
Pratik	1996

Query:

```
SELECT Name, Extract(SECOND FROM
BirthTime) AS BirthSecond FROM Test;
```

Output:

Name	BirthSecond
Pratik	581

DATE_ADD()

Adds a specified time interval to a date.

Syntax:

DATE_ADD(date, INTERVAL expr type);

Where, date – valid date expression, and expr is the number of intervals we want to add. and type can be one of the following: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc. Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL
1 YEAR) AS BirthTimeModified FROM Test;
```

Output:

Name	BirthTimeModified
Pratik	1997-09-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime,
INTERVAL 30 DAY) AS BirthDayModified FROM Test;
```

Output:

Name	BirthDayModified
Pratik	1996-10-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL
 4 HOUR) AS BirthHourModified FROM Test;
```

Output:

Name	BirthSecond
Pratik	1996-10-26 20:44:15.581

DATE_SUB()

Subtracts a specified time interval from a date. The syntax for DATE_SUB is the same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.

DATEDIFF()

Returns the number of days between two dates.

Syntax:

DATEDIFF(date1, date2);

date1 & date2- date/time expression

Query:

```
SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;
```

Output:

DateDiff
10

DATE_FORMAT()

Displays date/time data in different formats.

Syntax:

DATE_FORMAT(date,format);

the date is a valid date and the format specifies the output format for the date/time. The formats that can be used are:

- %a-Abbreviated weekday name (Sun-Sat)
- %b-Abbreviated month name (Jan-Dec)
- %c-Month, numeric (0-12)
- %D-Day of month with English suffix (0th, 1st, 2nd, 3rd)
- %d-Day of the month, numeric (00-31)
- %e-Day of the month, numeric (0-31)
- %f-Microseconds (000000-999999)
- %H-Hour (00-23)
- %h-Hour (01-12)
- %I-Hour (01-12)
- %i-Minutes, numeric (00-59)
- %j-Day of the year (001-366)
- %k-Hour (0-23)
- %l-Hour (1-12)
- %M-Month name (January-December)
- %m-Month, numeric (00-12)
- %p-AM or PM
- %r-Time, 12-hour (hh:mm: ss followed by AM or PM)
- %S-Seconds (00-59)
- %s-Seconds (00-59)
- %T-Time, 24-hour (hh:mm: ss)
- %U-Week (00-53) where Sunday is the first day of the week
- %u-Week (00-53) where Monday is the first day of the week
- %V-Week (01-53) where Sunday is the first day of the week, used with %X
- %v-Week (01-53) where Monday is the first day of the week, used with %x
- %W-Weekday name (Sunday-Saturday)
- %w-Day of the week (0=Sunday, 6=Saturday)
- %X-Year for the week where Sunday is the first day of the week, four digits, used with %V
- %x-Year for the week where Monday is the first day of the week, four digits, used with %v
- %Y-Year, numeric, four digits
- %y-Year, numeric, two digits

Last Updated : 04 Apr, 2023

33

Similar Reads

SQL | Date Functions (Set-1)

Sakshi98

In SQL, dates are complicated for newbies, since while working with a database, the format of the date in the table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

Some of the important date functions have been already discussed in the previous post. The basic idea of this post is to know the working or syntax of all the date functions:

Below are the date functions that are used in SQL:

1. **ADDDATE():** It returns a date after a certain time/date interval has been added.

   Syntax:
   ```
   SELECT ADDTIME("2018-07-16 02:52:47", "2");
   ```

   Output: 2018-07-16 02:52:49

2. **ADDTIME():** It returns a time / date time after a certain time interval has been added.

   Syntax:
   ```
   SELECT ADDTIME("2017-06-15 09:34:21", "2");
   ```

   Output: 2017-06-15 09:34:23

3. **CURDATE():** It returns the current date.

   Syntax:
   ```
   SELECT CURDATE();
   ```

   Output: 2018-07-16

4. **CURRENT_DATE():** It returns the current date.

   Syntax:
   ```
   SELECT CURRENT_DATE();
   ```

   Output: 2018-07-16

5. **CURRENT_TIME():** It returns the current time.

   Syntax:
   ```
   SELECT CURRENT_TIME();
   ```

   Output: 02:53:15

6. **CURRENT_TIMESTAMP():** It returns the current date and time.

   Syntax:
   ```
   SELECT CURRENT_TIMESTAMP();
   ```

Output: 2018-07-16 02:53:21

7. **CURTIME():** It returns the current time.

   Syntax: `SELECT CURTIME();`

   Output: 02:53:28

8. **DATE():** It extracts the date value from a date or date time expression.

   Syntax: `SELECT DATE("2017-06-15");`

   Output: 2017-06-15

9. **DATEDIFF():** It returns the difference in days between two date values.

   Syntax: `SELECT DATEDIFF("2017-06-25", "2017-06-15");`

   Output: 10

10. **DATE_ADD():** It returns a date after a certain time/date interval has been added.

    Syntax: `SELECT DATE_ADD("2018-07-16", INTERVAL 10 DAY);`

    Output: 2018-07-16

11. **DATE_FORMAT():** It formats a date as specified by a format mask.

    Syntax: `SELECT DATE_FORMAT("2018-06-15", "%Y");`

    Output: 2018

12. **DATE_SUB():** It returns a date after a certain time/date interval has been subtracted.

    Syntax: `SELECT DATE_SUB("2017-06-15", INTERVAL 10 DAY);`

    Output: 2018-07-16

13. **DAY():** It returns the day portion of a date value.

    Syntax: `SELECT DAY("2018-07-16");`

    Output: 16

14. **DAYNAME():** It returns the weekday name for a date.

    Syntax: `SELECT DAYNAME('2008-05-15');`

    Output: Thursday

15. **DAYOFMONTH():** It returns the day portion of a date value.

    Syntax: `SELECT DAYOFMONTH('2018-07-16');`

    Output: 16

16. **DAYWEEK():** It returns the weekday index for a date value.

Syntax: `SELECT WEEKDAY("2018-07-16");`

Output: 0

17. **DAYOFYEAR():** It returns the day of the year for a date value.

    Syntax: `SELECT DAYOFYEAR("2018-07-16");`

    Output: 197

18. **EXTRACT():** It extracts parts from a date.

    Syntax: `SELECT EXTRACT(MONTH FROM "2018-07-16");`

    Output: 7

19. **FROM_DAYS():** It returns a date value from a numeric representation of the day.

    Syntax: `SELECT FROM_DAYS(685467);`

    Output: 1876-09-29

20. **HOUR():** It returns the hour portion of a date value.

    Syntax: `SELECT HOUR("2018-07-16 09:34:00");`

    Output: 9

21. **LAST_DAY():** It returns the last day of the month for a given date.

    Syntax: `SELECT LAST_DAY('2018-07-16');`

    Output: 2018-07-31

22. **LOCALTIME():** It returns the current date and time.

    Syntax: `SELECT LOCALTIME();`

    Output: 2018-07-16 02:56:42

23. **LOCALTIMESTAMP():** It returns the current date and time.

    Syntax: `SELECT LOCALTIMESTAMP();`

    Output: 2018-07-16 02:56:48

24. **MAKEDATE():** It returns the date for a certain year and day-of-year value.

    Syntax: `SELECT MAKEDATE(2009, 138);`

    Output: 2009-05-18

25. **MAKETIME():** It returns the time for a certain hour, minute, second combination.

    Syntax: `SELECT MAKETIME(11, 35, 4);`

SQL | Date Functions (Set-2)

Sakshi98

In SQL, dates are complicated for newbies, since while working with a database, the format of the date in the table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

Some of the date functions have been already discussed in the Set-1. In this post, the remaining date functions have been discussed.

Below are the remaining date functions that are used in SQL:

1. **MICROSECOND():** It returns the microsecond portion of a date value.

   Syntax: `SELECT MICROSECOND("2018-07-18 09:12:00.000345");`

   Output: 345

2. **MINUTE():** It returns the minute portion of a date value.

   Syntax: `SELECT MINUTE("2018-07-18 09:12:00");`

   Output: 12

3. **MONTH():** It returns the month portion of a date value.

   Syntax: `SELECT MONTH ('2018/07/18')AS MONTH;`

   Output: 7

4. **MONTHNAME():** It returns the full month name for a date.

   Syntax: `SELECT MONTHNAME("2018/07/18");`

   Output: JULY

5. **NOW():** It returns the current date and time.

   Syntax: `SELECT NOW();`

   Output: 2018-07-18 09:14:32

6. **PERIOD_ADD():** It takes a period and adds a specified number of months to it.

   Syntax: `SELECT PERIOD_ADD(201803, 6);`

Output: 201809

7. **PERIOD_DIFF():** It returns the difference in months between two periods.

   Syntax: `SELECT PERIOD_DIFF(201810, 201802);`

   Output: 8

8. **QUARTER():** It returns the quarter portion of a date value.

   Syntax: `SELECT QUARTER("2018/07/18");`

   Output: 3

9. **SECOND():** It returns the second portion of a date value.

   Syntax: `SELECT SECOND("09:14:00:00032");`

   Output: 0

10. **SEC_TO_TIME():** It converts numeric seconds into a time value.

    Syntax: `SELECT SEC_TO_TIME(1);`

    Output: 00:00:01

11. **STR_TO_DATE():** It takes a string and returns a date specified by a format mask.

    Syntax: `SELECT STR_TO_DATE("JULY 18 2018", "%M %D %Y");`

    Output: 0018-07-18

12. **SUBDATE():** It returns a date after which a certain time/date interval has been subtracted.

    Syntax: `SELECT SUBDATE("2017-06-15", INTERVAL 10 DAY);`

    Output: 2017-06-05

13. **SUBTIME():** It returns a time/date time value after a certain time interval has been subtracted.

    Syntax: `SELECT SUBDATE("2018/07/18", INTERVAL 10 DAY);`

    Output: 2018-07-18 09:15:17.542768

14. **SYSDATE():** It returns the current date and time.

    Syntax: `SELECT SYSDATE();`

    Output: 2018-07-18 09:19:03

15. **TIME():** It extracts the time value from a time/date time expression.

    Syntax: `SELECT TIME("09:16:10");`

    Output: 09:16:10

16. **TIME_FORMAT():** It formats the time as specified by a format mask.

    Syntax: `SELECT TIME_FORMAT("09:16:10", "%H %I %S");`

    Output: 09 09 10

17. **TIME_TO_SEC():** It converts a time value into numeric seconds.

    Syntax: `SELECT TIME_TO_SEC("09:16:10");`

    Output: 33370

18. **TIMEDIFF():** It returns the difference between two time/datetime values.

    Syntax: `SELECT TIMEDIFF("09:16:10", "09:16:04");`

    Output: 00:00:06

19. **TIMESTAMP():** It converts an expression to a date time value and if specified adds an optional time interval to the value.

    Syntax: `SELECT TIMESTAMP("2018-07-18", "09:16:10");`

    Output: 2018-07-18 09:16:10

20. **TO_DAYS():** It converts a date into numeric days.

    Syntax: `SELECT TO_DAYS("2018-07-18");`

    Output: 737258

21. **WEEK():** It returns the week portion of a date value.

    Syntax: `SELECT WEEK("2018-07-18");`

    Output: 28

22. **WEEKDAY():** It returns the weekday index for a date value.

    Syntax: `SELECT WEEKDAY("2018-07-18");`

    Output: 2

23. **WEEKOFYEAR():** It returns the week of the year for a date value.

    Syntax: `SELECT WEEKOFYEAR("2018-07-18");`

    Output: 29

24. **YEAR():** It returns the year portion of a date value.

    Syntax: `SELECT YEAR("2018-07-18");`

    Output: 2018

25. **YEARWEEK():** It returns the year and week for a date value.

Syntax: `SELECT YEARWEEK("2018-07-18");`

Output: 201828

Last Updated : 19 Jul, 2018

4

Similar Reads

1. SQL | Date Functions (Set-1)
2. How to Write a SQL Query For a Specific Date Range and Date Time?
3. SQL | Date functions
4. Useful Date and Time Functions in PL/SQL
5. Date manipulating functions in SQL
6. SQL Query to Check if Date is Greater Than Today in SQL
7. SQL | Functions (Aggregate and Scalar Functions)
8. SQL | Difference between functions and stored procedures in PL/SQL
9. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
10. Configure SQL Jobs in SQL Server using T-SQL

Previous Next

Article Contributed By :

Sakshi98
Sakshi98

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Article Tags : SQL-Functions, SQL

Practice Tags : SQL

SQL | LISTAGG

LISTAGG function in DBMS is used to aggregate strings from data in columns in a database table.

- It makes it very easy to concatenate strings. It is similar to concatenation but uses grouping.
- The speciality about this function is that, it also allows to order the elements in the concatenated list.

Syntax:

```
LISTAGG (measure_expr [, 'delimiter']) WITHIN GROUP
(order_by_clause) [OVER query_partition_clause]
measure_expr : The column or expression to concatenate the values.
delimiter : Character in between each measure_expr, which is by default a comma
(,) .
order_by_clause : Order of the concatenated values.
```

Let us have a table named Gfg having two columns showing the subject names and subject number that each subject belongs to, as shown below :

```
SQL> select * from GfG;

SUBNO      SUBNAME
---------- ------------------------------
D20        Algorithm
D30        DataStructure
D30        C
D20        C++
D30        Python
D30        DBMS
D10        LinkedList
D20        Matrix
D10        String
D30        Graph
D20        Tree
```

```
11 rows selected.
```

Query 1: Write an SQL query using LISTAGG function to output the subject names in a single field with the values comma delimited.


```
SQL> SELECT LISTAGG(SubName, ' , ') WITHIN GROUP (ORDER BY SubName) AS SUBJECTS
  2  FROM   GfG ;
```

Output:

```
SUBJECTS
--------------------------------------------------------------------------------
--
Algorithm , C , C++ , DBMS , DataStructure , Graph , LinkedList , Matrix , Python
,
String , Tree
```

Query 2: Write an SQL query to group each subject and show each subject in its respective department separated by comma with the help of LISTAGG function.

```
SQL> SELECT SubNo, LISTAGG(SubName, ' , ') WITHIN GROUP (ORDER BY SubName) AS
SUBJECTS
  2  FROM   GfG
  3  GROUP BY SubNo;
```

Output:

```
SUBNO      SUBJECTS
------     --------------------------------------------------------------------
----------
D10        LinkedList , String
D20         Algorithm , C++ , Matrix , Tree
D30         C , DBMS , DataStructure , Graph , Python
```

Query 3: Write an SQL query to show the subjects belonging to each department ordered by the subject number (SUBNO) with the help of LISTAGG function.

```
SQL> SELECT SubNo, LISTAGG(SubName, ',') WITHIN GROUP (ORDER BY SubName) AS
SUBJECTS
  2  FROM   GfG
  3  GROUP BY SubNo
  4  ORDER BY SubNo;
```

Output:

```
SUBNO        SUBJECTS
-----        --------------------------------
D10          LinkedList, String
D20          Algorithm, C++, Matrix, Tree
D30          C, DBMS, DataStructure, Graph, Python
```

This article is contributed by MAZHAR IMAM KHAN. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 22 Jun, 2018

12

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Configure SQL Jobs in SQL Server using T-SQL
3. SQL | Procedures in PL/SQL
4. SQL | Difference between functions and stored procedures in PL/SQL
5. SQL SERVER – Input and Output Parameter For Dynamic SQL
6. Difference between SQL and T-SQL
7. SQL Server | Convert tables in T-SQL into XML
8. SQL SERVER | Bulk insert data from csv file using T-SQL command
9. SQL - SELECT from Multiple Tables with MS SQL Server

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

1) Count()
2) Sum()
3) Avg()
4) Min()
5) Max()

Now let us understand each Aggregate function with a example:

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

Count():

Count(*): Returns total number of records .i.e 6.
Count(salary): Return number of Non Null values over the column salary. i.e 5.
Count(Distinct Salary): Return number of distinct Non Null values over the column salary .i.e 4

Sum():

sum(salary): Sum all Non Null values of Column salary i.e., 310
sum(Distinct salary): Sum of all distinct Non-Null values i.e., 250.

Avg():

Avg(salary) = Sum(salary) / count(salary) = 310/5
Avg(Distinct salary) = sum(Distinct salary) / Count(Distinct Salary) = 250/4

Min():

Min(salary): Minimum value in the salary column except NULL i.e., 40.
Max(salary): Maximum value in the salary i.e., 80.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 20 Aug, 2019

115

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)
2. Aggregate functions in Cassandra
3. SQL | Difference between functions and stored procedures in PL/SQL
4. SQL vs NO SQL vs NEW SQL

SQL | Functions (Aggregate and Scalar Functions)

For doing operations on data SQL has many built-in functions, they are categorized in two categories and further sub-categorized in different seven functions under each category. The categories are:

1. **Aggregate functions:**
   These functions are used to do operations from the values of the column and a single value is returned.
   1. AVG()
   2. COUNT()
   3. FIRST()
   4. LAST()
   5. MAX()
   6. MIN()
   7. SUM()

2. **Scalar functions:**
   These functions are based on user input, these too returns single value.
   1. UCASE()
   2. LCASE()
   3. MID()
   4. LEN()
   5. ROUND()
   6. NOW()
   7. FORMAT()

Students-Table

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Aggregate Functions

AVG(): It returns the average value after calculating from values in a numeric column.

Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Queries:

- Computing average marks of students.

```
SELECT AVG(MARKS) AS AvgMarks FROM Students;
```

Output:

AvgMarks
80

- Computing average age of students.

```
SELECT AVG(AGE) AS AvgAge FROM Students;
```

Output:

AvgAge
19.4

COUNT(): It is used to count the number of rows returned in a SELECT statement. It can't be used in MS ACCESS.

Syntax:

```
SELECT COUNT(column_name) FROM table_name;
```

Queries:

- Computing total number of students.

```
SELECT COUNT(*) AS NumStudents FROM Students;
```

Output:

NumStudents
5

- Computing number of students with unique/distinct age.

```
SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;
```

Output:

NumStudents
4

FIRST(): The FIRST() function returns the first value of the selected column.

Syntax:

```
SELECT FIRST(column_name) FROM table_name;
```

Queries:

- Fetching marks of first student from the Students table.

```
SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

Output:

MarksFirst
90

- Fetching age of first student from the Students table.

```
SELECT FIRST(AGE) AS AgeFirst FROM Students;
```

Output:

AgeFirst
19

LAST(): The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.

Syntax:

```
SELECT LAST(column_name) FROM table_name;
```

Queries:

- Fetching marks of last student from the Students table.

```
SELECT LAST(MARKS) AS MarksLast FROM Students;
```

Output:

MarksLast
85

- Fetching age of last student from the Students table.

```
SELECT LAST(AGE) AS AgeLast FROM Students;
```

Output:

AgeLast
18

MAX(): The MAX() function returns the maximum value of the selected column.

Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

Queries:

- Fetching maximum marks among students from the Students table.

```
SELECT MAX(MARKS) AS MaxMarks FROM Students;
```

Output:

MaxMarks
95

- Fetching max age among students from the Students table.

```
SELECT MAX(AGE) AS MaxAge FROM Students;
```

Output:

MaxAge
21

MIN(): The MIN() function returns the minimum value of the selected column.

Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Queries:

- Fetching minimum marks among students from the Students table.

```
SELECT MIN(MARKS) AS MinMarks FROM Students;
```

Output:

MinMarks
50

- Fetching minimum age among students from the Students table.

```
SELECT MIN(AGE) AS MinAge FROM Students;
```

Output:

MinAge
18

SUM(): The SUM() function returns the sum of all the values of the selected column.

Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Queries:

- Fetching summation of total marks among students from the Students table.

```
SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

Output:

TotalMarks
400

- Fetching summation of total age among students from the Students table.

```
SELECT SUM(AGE) AS TotalAge FROM Students;
```

Output:

TotalAge
97

Scalar Functions

UCASE(): It converts the value of a field to uppercase.

Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Queries:

- Converting names of students from the table Students to uppercase.

```
SELECT UCASE(NAME) FROM Students;
```

Output:

NAME
HARSH
SURESH
PRATIK
DHANRAJ
RAM

LCASE(): It converts the value of a field to lowercase.

Syntax:

```
SELECT LCASE(column_name) FROM table_name;
```

Queries:

- Converting names of students from the table Students to lowercase.

```
SELECT LCASE(NAME) FROM Students;
```

Output:

NAME
harsh
suresh
pratik
dhanraj
ram

MID(): The MID() function extracts texts from the text field.

Syntax:

```
SELECT MID(column_name,start,length) AS some_name FROM table_name;

specifying length is optional here, and start signifies start position ( starting
from 1 )
```

Queries:

- Fetching first four characters of names of students from the Students table.

```
SELECT MID(NAME,1,4) FROM Students;
```

Output:

NAME
HARS
SURE

PRAT
DHAN
RAM

LEN(): The LEN() function returns the length of the value in a text field.

Syntax:

```
SELECT LENGTH(column_name) FROM table_name;
```

Queries:

- Fetching length of names of students from Students table.

```
SELECT LENGTH(NAME) FROM Students;
```

Output:

NAME
5
6
6
7
3

ROUND(): The ROUND() function is used to round a numeric field to the number of decimals specified.NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;

decimals- number of decimals to be fetched.
```

Queries:

- Fetching maximum marks among students from the Students table.

```
SELECT ROUND(MARKS,0) FROM table_name;
```

Output:

MARKS
90
50
80
95
85

NOW(): The NOW() function returns the current system date and time.

Syntax:

```
SELECT NOW() FROM table_name;
```

Queries:

- Fetching current system time.

```
SELECT NAME, NOW() AS DateTime FROM Students;
```

Output:

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM
PRATIK	1/13/2017 1:30:11 PM

DHANRAJ	1/13/2017 1:30:11 PM
RAM	1/13/2017 1:30:11 PM

FORMAT(): The FORMAT() function is used to format how a field is to be displayed.

Syntax:

```
SELECT FORMAT(column_name,format) FROM table_name;
```

Queries:

- Formatting current date as 'YYYY-MM-DD'.

```
SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;
```

Output:

NAME	Date
HARSH	2017-01-13
SURESH	2017-01-13
PRATIK	2017-01-13
DHANRAJ	2017-01-13
RAM	2017-01-13

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 08 Jul, 2022

42

Similar Reads

SQL | NULL functions

In the database, null values serve as placeholders for data that is either missing or not available. a null value is a flexible data type that can be placed in the column of any data type, including string, int, blob, and CLOB datatypes. It is not a component of any specific data type. Null values are helpful when cleaning the data prior to exploratory analysis.

Null values assist us in eradicating data ambiguity. Null values are also useful for maintaining a consistent datatype across the column. We will learn about the necessity and guidelines for using Null values in this article. Now let's use examples to try to better understand null values and null functions in SQL.

Why do We Need NULL Values?

Null functions are required to perform operations on null values stored in the database. With NULL values, we can perform operations that clearly identify whether the value is null or not. With this ability to recognize null data, operations similar to SQL's join methods can be performed on them.

Following are the NULL functions defined in SQL:

ISNULL()

The ISNULL function has different uses in SQL Server and MySQL. In SQL Server, ISNULL() function is used to replace NULL values.

Syntax:

```
SELECT column(s), ISNULL(column_name, value_to_replace)
```

FROM table_name;

Example: Consider the following Employee table,

Name	Salary
John	8000
William	NULL

Find the sum of the salary of all Employees, if the Salary of any employee is not available (or NULL value), use salary as 10000.

Query:

```
SELECT SUM(ISNULL(Salary, 10000) AS Salary
FROM Employee;
```

Output:

Salary
18000

In MySQL, ISNULL() function is used to test whether an expression is NULL or not. If the expression is NULL it returns TRUE, else FALSE.

Syntax:

SELECT column(s)

FROM table_name

WHERE ISNULL(column_name);

Example: Consider the following Employee table

Name	Salary
John	8000
William	NULL

Fetch the name of all employees whose salary is available in the table (not NULL).

Query:

```
SELECT Name
FROM Employee
WHERE ISNULL(Salary);
```

Output:

Name
William

IFNULL()

This function is available in MySQL, and not in SQL Server or Oracle. This function take two arguments. If the first argument is not NULL, the function returns the first argument. Otherwise, the second argument is returned. This function is commonly used to replace NULL value with another value.

Syntax:

SELECT column(s), IFNULL(column_name, value_to_replace)

FROM table_name;

Example: Consider the following Employee table,

Name	Salary
John	8000
William	NULL

Find the sum of the salary of all Employees, if the Salary of any employee is not available (or NULL value), use salary as 10000.

Query;

```
SELECT SUM(IFNULL(Salary, 10000) AS Salary
FROM Employee;
```

Output:

Salary
18000

COALESCE()

COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.

Syntax:

SELECT column(s), CAOLESCE(expression_1,....,expression_n)

FROM table_name;

Example:

Consider the following Contact_info table,

Name	Phone1	Phone2
John	1234567897	1258741235
William	NULL	7897894561

Fetch the name and contact number of each employee.

Query:

```
SELECT Name, COALESCE(Phone1, Phone2) AS Contact
FROM Contact_info;
```

Output:

Name	Contact
John	1258741235
William	7897894561

NULLIF()

The NULLIF function takes two arguments. If the two arguments are equal, then NULL is returned. Otherwise, the first argument is returned.

Syntax:

SELECT column(s), NULLIF(expression1, expression2)

FROM table_name;

Example: Consider the following Sales table

Store	Actual	Goal
Store_A	50	50
Store_B	80	100

```
SELECT Store, NULLIF(Actual, Goal)
FROM Sales;
```

Output:

Store	NULLIF (Actual, Goal)
Store_A	NULL
Store_B	80

Conclusion

In this article, we learned what null values are and why we should use them. We now know that using NULL values is fundamental to databases and is done so in order to preserve their integrity. Following this, we learned more about the different functions that can be used with NULL values.

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 22 May, 2023

3

Similar Reads

1. How to Alter a Column from Null to Not Null in SQL Server?

SQL | Numeric Functions

Sakshi98

Numeric Functions are used to perform operations on numbers and return numbers. Following are the numeric functions defined in SQL:

ABS(): It returns the absolute value of a number.

```
Syntax: SELECT ABS(-243.5);
```

Output: 243.5

```
SQL> SELECT ABS(-10);
+-------------------------------------+
| ABS(10)
+-------------------------------------+
| 10
+-------------------------------------+
```

ACOS(): It returns the cosine of a number, in radians.


```
Syntax: SELECT ACOS(0.25);
```

Output: 1.318116071652818

ASIN(): It returns the arc sine of a number, in radians.

```
Syntax: SELECT ASIN(0.25);
```

Output: 0.25268025514207865

ATAN(): It returns the arc tangent of a number, in radians.

```
Syntax: SELECT ATAN(2.5);
```

Output: 1.1902899496825317

CEIL(): It returns the smallest integer value that is greater than or equal to a number.

```
Syntax: SELECT CEIL(25.75);
```

Output: 26

CEILING(): It returns the smallest integer value that is greater than or equal to a number.

```
Syntax: SELECT CEILING(25.75);
```

Output: 26

COS(): It returns the cosine of a number, in radians.

```
Syntax: SELECT COS(30);
```

Output: 0.15425144988758405

COT(): It returns the cotangent of a number, in radians.

```
Syntax: SELECT COT(6);
```

Output: -3.436353004180128

DEGREES(): It converts a radian value into degrees.

```
Syntax: SELECT DEGREES(1.5);
```

Output: 85.94366926962348

```
SQL>SELECT DEGREES(PI());
+-----------------------------------------+
| DEGREES(PI())
+-----------------------------------------+
| 180.000000
+-----------------------------------------+
```

DIV(): It is used for integer division.

```
Syntax: SELECT 10 DIV 5;
```

Output: 2

EXP(): It returns e raised to the power of a number.

```
Syntax: SELECT EXP(1);
```

Output: 2.718281828459045

FLOOR(): It returns the largest integer value that is less than or equal to a number.

```
Syntax: SELECT FLOOR(25.75);
```

Output: 25

GREATEST(): It returns the greatest value in a list of expressions.

```
Syntax: SELECT GREATEST(30, 2, 36, 81, 125);
```

Output: 125

LEAST(): It returns the smallest value in a list of expressions.

```
Syntax: SELECT LEAST(30, 2, 36, 81, 125);
```

Output: 2

LN(): It returns the natural logarithm of a number.

```
Syntax: SELECT LN(2);
```

Output: 0.6931471805599453

LOG10(): It returns the base-10 logarithm of a number.

```
Syntax: SELECT LOG(2);
```

Output: 0.6931471805599453

LOG2(): It returns the base-2 logarithm of a number.

```
Syntax: SELECT LOG2(6);
```

Output: 2.584962500721156

MOD(): It returns the remainder (aka. modulus) of n divided by m.

```
Syntax: SELECT MOD(18, 4);
```

Output: 2

PI(): It returns the value of Pi and displays 6 decimal places.

```
Syntax: SELECT PI();
```

Output: 3.141593

POWER(m, n): It returns m raised to the nth power.

```
Syntax: SELECT POWER(4, 2);
```

Output: 16

RADIANS(): It converts a value in degrees to radians.

```
Syntax: SELECT RADIANS(180);
```

Output: 3.141592653589793

RAND(): It returns a random number between 0 (inclusive) and 1 (exclusive).

```
Syntax: SELECT RAND();
```

Output: 0.33623238684258644

ROUND(): It returns a number rounded to a certain number of decimal places.

```
Syntax: SELECT ROUND(5.553);
```

Output: 6

SIGN(): It returns a value indicating the sign of a number. A return value of 1 means positive; 0 means negative.

```
Syntax: SELECT SIGN(255.5);
```

Output: 1

SIN(): It returns the sine of a number in radians.

```
Syntax: SELECT SIN(2);
```

Output: 0.9092974268256817

SQRT(): It returns the square root of a number.

```
Syntax: SELECT SQRT(25);
```

Output: 5

TAN(): It returns the tangent of a number in radians.

```
Syntax: SELECT TAN(1.75);
```

Output: -5.52037992250933

ATAN2(): It returns the arctangent of the x and y coordinates, as an angle and expressed in radians.

```
Syntax: SELECT ATAN2(7);
```

Output: 1.42889927219073

TRUNCATE(): This doesn't work for SQL Server. It returns 7.53635 truncated to n places right of the decimal point.

```
Syntax: SELECT TRUNCATE(7.53635, 2);
```

Output: 7.53

Last Updated : 12 Feb, 2023

11

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)
2. SQL | Difference between functions and stored procedures in PL/SQL
3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
4. Configure SQL Jobs in SQL Server using T-SQL
5. Various String, Numeric, and Date & Time functions in MySQL
6. Numeric and Date-time data types in SQL Server
7. PHP - My SQL : abs() - numeric function
8. SQL Query to convert NUMERIC to NVARCHAR
9. SQL | Date functions
10. SQL | NULL functions

Previous Next

SQL | String functions

Sakshi98

String functions
are used to perform an operation on input string and return an output string.
Following are the string functions defined in SQL:

1. **ASCII():** This function is used to find the ASCII value of a character.

   ```
   Syntax: SELECT ascii('t');
   Output: 116
   ```

2. **CHAR_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a word.

   ```
   Syntax: SELECT char_length('Hello!');
   Output: 6
   ```

3. **CHARACTER_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a line.

   ```
   Syntax: SELECT CHARACTER_LENGTH('geeks for geeks');
   Output: 15
   ```

4. **CONCAT():** This function is used to add two words or strings.

   ```
   Syntax: SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;
   Output: 'GeeksforGeeks'
   ```

5. **CONCAT_WS():** This function is used to add two words or strings with a symbol as concatenating symbol.

   ```
   Syntax: SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');
   Output: geeks_for_geeks
   ```

6. **FIND_IN_SET():** This function is used to find a symbol from a set of symbols.

   ```
   Syntax: SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');
   Output: 2
   ```

7. **FORMAT():** This function is used to display a number in the given format.

Syntax: Format("0.981", "Percent");
Output: ‘98.10%’

8. **INSERT():** This function is used to insert the data into a database.

   Syntax: INSERT INTO database (geek_id, geek_name) VALUES (5000, 'abc');
   Output: successfully updated

9. **INSTR():** This function is used to find the occurrence of an alphabet.

   Syntax: INSTR('geeks for geeks', 'e');
   Output: 2 (the first occurrence of ‘e’)

   Syntax: INSTR('geeks for geeks', 'e', 1, 2);
   Output: 3 (the second occurrence of ‘e’)

10. **LCASE():** This function is used to convert the given string into lower case.

    Syntax: LCASE ("GeeksFor Geeks To Learn");
    Output: geeksforgeeks to learn

11. **LEFT():** This function is used to SELECT a sub string from the left of given size or characters.

    Syntax: SELECT LEFT('geeksforgeeks.org', 5);
    Output: geeks

12. **LENGTH():** This function is used to find the length of a word.

    Syntax: LENGTH('GeeksForGeeks');
    Output: 13

13. **LOCATE():** This function is used to find the nth position of the given word in a string.

    Syntax: SELECT LOCATE('for', 'geeksforgeeks', 1);
    Output: 6

14. **LOWER():** This function is used to convert the upper case string into lower case.

    Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');
    Output: geeksforgeeks.org

15. **LPAD():** This function is used to make the given string of the given size by adding the given symbol.

    Syntax: LPAD('geeks', 8, '0');
    Output:
    000geeks

16. **LTRIM():** This function is used to cut the given sub string from the original string.

```
Syntax: LTRIM('123123geeks', '123');
Output: geeks
```

17. **MID():** This function is to find a word from the given position and of the given size.

```
Syntax: Mid ("geeksforgeeks", 6, 2);
Output: for
```

18. **POSITION():** This function is used to find position of the first occurrence of the given alphabet.

```
Syntax: SELECT POSITION('e' IN 'geeksforgeeks');
Output: 2
```

19. **REPEAT():** This function is used to write the given string again and again till the number of times mentioned.

```
Syntax: SELECT REPEAT('geeks', 2);
Output: geeksgeeks
```

20. **REPLACE():** This function is used to cut the given string by removing the given sub string.

```
Syntax: REPLACE('123geeks123', '123');
Output: geeks
```

21. **REVERSE():** This function is used to reverse a string.

```
Syntax: SELECT REVERSE('geeksforgeeks.org');
Output: ‘gro.skeegrofskeeg’
```

22. **RIGHT():** This function is used to SELECT a sub string from the right end of the given size.

```
Syntax: SELECT RIGHT('geeksforgeeks.org', 4);
Output: ‘.org’
```

23. **RPAD():** This function is used to make the given string as long as the given size by adding the given symbol on the right.

```
Syntax: RPAD('geeks', 8, '0');
Output: ‘geeks000’
```

24. **RTRIM():** This function is used to cut the given sub string from the original string.

```
Syntax: RTRIM('geeksxyxzyyy', 'xyz');
Output: ‘geeks’
```

25. **SPACE():** This function is used to write the given number of spaces.

```
Syntax: SELECT SPACE(7);
Output: ‘       ‘
```

26. **STRCMP():** This function is used to compare 2 strings.
    - If string1 and string2 are the same, the STRCMP function will return 0.
    - If string1 is smaller than string2, the STRCMP function will return -1.
    - If string1 is larger than string2, the STRCMP function will return 1.

```
Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');
Output: -1
```

27. **SUBSTR():** This function is used to find a sub string from the a string from the given position.

```
Syntax:SUBSTR('geeksforgeeks', 1, 5);
Output: ‘geeks’
```

28. **SUBSTRING():** This function is used to find an alphabet from the mentioned size and the given string.

```
Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);
Output: ‘G’
```

29. **SUBSTRING_INDEX():** This function is used to find a sub string before the given symbol.

```
Syntax: SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);
Output: ‘www’
```

30. **TRIM():** This function is used to cut the given symbol from the string.

```
Syntax: TRIM(LEADING '0' FROM '000123');
Output: 123
```

31. **UCASE():** This function is used to make the string in upper case.

```
Syntax: UCASE ("GeeksForGeeks");
Output:
GEEKSFORGEEKS
```

Last Updated : 30 Dec, 2019

29

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)
2. SQL | Difference between functions and stored procedures in PL/SQL
3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
4. Configure SQL Jobs in SQL Server using T-SQL

SQL | Advanced Functions

Sakshi98

SQL (Structured Query Language) offers a wide range of advanced functions that allow you to perform complex calculations, transformations, and aggregations on your data.

Aggregate Functions

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

- SUM(): Calculates the sum of values in a column.
- AVG(): Computes the average of values in a column.
- COUNT(): Returns the number of rows or non-null values in a column.
- MIN(): Finds the minimum value in a column.
- MAX(): Retrieves the maximum value in a column.

Conditional Functions

- CASE WHEN: Allows conditional logic to be applied in the SELECT statement.
- COALESCE(): Returns the first non-null value in a list.
- NULLIF(): Compares two expressions and returns null if they are equal; otherwise, returns the first expression.

Mathematical Functions

Mathematical functions are present in SQL which can be used to perform mathematical calculations. Some commonly used mathematical functions are given below:

- ABS(): Returns the absolute value of a number.
- ROUND(): Rounds a number to a specified number of decimal places.
- POWER(): Raises a number to a specified power.
- SQRT(): Calculates the square root of a number.

Advanced Functions in SQL

BIN(): It converts a decimal number to a binary number.

Query:

```
SELECT BIN(18);
```

Output:

BIN(18)
10010

BINARY(): It converts a value to a binary string.

Query:

```
SELECT BINARY "GeeksforGeeks";
```

Output:

BINARY "GeeksforGeeeks"
GeeksforGeeks

COALESCE(): It returns the first non-null expression in a list.

Query:

```
SELECT COALESCE(NULL,NULL,'GeeksforGeeks',NULL,'Geeks');
```

Output:

COALESCE(NULL,NULL,'GeeksforGeeks',NULL,'Geeks')
GeeksforGeeks

CONNECTION_ID(): It returns the unique connection ID for the current connection.

Query:

```
SELECT CONNECTION_ID();
```

Output:

CONNECTION_ID()
9

CURRENT_USER(): It returns the user name and hostname for the MySQL account used by the server to authenticate the current client.

Query:

```
SELECT CURRENT_USER();
```

Output:

CURRENT_USER()
root@localhost

DATABASE(): It returns the name of the default database.

Query:

```
SELECT DATABASE();
```

Output:

DATABASE()
NULL

IF(): It returns one value if a condition is TRUE, or another value if a condition is FALSE.

Query:

```
SELECT IF(200<500, "YES", "NO");
```

Output:

IF(200<500, "YES", "NO")
YES

LAST_INSERT_ID(): It returns the first AUTO_INCREMENT value that was set by the most recent INSERT or UPDATE statement.

Query:

```
SELECT LAST_INSERT_ID();
```

Output:

LAST_INSERT_ID()
0

Query:

```
SELECT NULLIF(25.11, 25);
```

Output:

NULLIF(25.11, 25)
25.11

Query:

```
SELECT NULLIF(115, 115);
```

Output:

NULLIF(115, 115)
NULL

SESSION_USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT SESSION_USER();
```

Output:

SESSION_USER()
root@localhost

SYSTEM_USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT SYSTEM_USER();
```

Output:

SYSTEM_USER()
root@localhost

USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT USER();
```

Output:

USER()
root@localhost

VERSION(): It returns the version of the MySQL database.

Query:

```
SELECT VERSION();
```

Output:

VERSION()
8.0.11

Last Updated : 26 May, 2023

6

Similar Reads

SQL | Subquery

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query. Important rules for Subqueries:

- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as **main query** and inner query is called as **subquery**.
- The subquery generally executes first when the subquery doesn't have any **co-relation** with the **main query**, when there is a co-relation the parser takes the decision **on the fly** on which query to execute on **precedence** and uses the output of the subquery accordingly.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command **cannot** be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

Syntax: There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
    ( SELECT COLUMN_NAME  from TABLE_NAME   WHERE ... );
```

Sample Table:

DATABASE

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

STUDENT

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

Sample Queries

:

- To display NAME, LOCATION, PHONE_NUMBER of the students from DATABASE table whose section is A

```
Select NAME, LOCATION, PHONE_NUMBER from DATABASE
WHERE ROLL_NO IN
(SELECT ROLL_NO from STUDENT where SECTION='A');
```

- **Explanation :** First subquery executes " SELECT ROLL_NO from STUDENT where SECTION='A' " returns ROLL_NO from STUDENT table whose SECTION is 'A'.Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner subquery. Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER

Ravi	104	Salem	8989898989
Raj	102	Coimbatore	8877665544

- Insert Query Example:

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	**101**	**chennai**	**9988773344**
Raju	**102**	**coimbatore**	**9090909090**
Ravi	**103**	**salem**	**8989898989**

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	**111**	**chennai**	**8787878787**
Sai	**112**	**mumbai**	**6565656565**
Sri	**113**	**coimbatore**	**7878787878**

- To insert Student2 into Student1 table:

```
INSERT INTO Student1  SELECT * FROM Student2;
```

- Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	**101**	**chennai**	**9988773344**
Raju	**102**	**coimbatore**	**9090909090**
Ravi	**103**	**salem**	**8989898989**
Raj	**111**	**chennai**	**8787878787**
Sai	**112**	**mumbai**	**6565656565**
Sri	**113**	**coimbatore**	**7878787878**

- To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai

```
DELETE FROM Student2
WHERE ROLL_NO IN ( SELECT ROLL_NO
                   FROM Student1
                   WHERE LOCATION = 'chennai');
```

- Output:

```
1 row delete successfully.
```

- **Display Student2 table:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table

```
UPDATE Student2
SET NAME='geeks'
WHERE LOCATION IN ( SELECT LOCATION
                    FROM Student1
                    WHERE NAME IN ('Raju','Ravi'));
```

- Output:

```
1 row updated successfully.
```

- **Display Student2 table:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
geeks	113	coimbatore	7878787878

This article is contributed by **RanjaniRavi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

SQL | Sub queries in From Clause

Mayank Kumar 12

From clause can be used to specify a sub-query expression in SQL. The relation produced by the sub-query is then used as a new relation on which the outer query is applied.

- Sub queries in the from clause are supported by most of the SQL implementations.
- The correlation variables from the relations in from clause cannot be used in the sub-queries in the from clause.

Syntax:

```
SELECT column1, column2 FROM
(SELECT column_x  as C1, column_y FROM table WHERE PREDICATE_X)
as table2, table1
WHERE PREDICATE;
```

Note: The sub-query in the from clause is evaluated first and then the results of evaluation are stored in a new temporary relation.
Next, the outer query is evaluated, selecting only those tuples from the temporary relation that satisfies the predicate in the where clause of the outer query.

Query

Example 1: Find all professors whose salary is greater than the average budget of all the departments.

Instructor relation:

InstructorID	Name	Department	Salary
44547	Smith	Computer Science	95000
44541	Bill	Electrical	55000
47778	Sam	Humanities	44000
48147	Erik	Mechanical	80000
411547	Melisa	Information Technology	65000
48898	Jena	Civil	50000

Department relation:

Department Name	Budget
Computer Science	100000
Electrical	80000
Humanities	50000
Mechanical	40000
Information Technology	90000
Civil	60000

Query:

```
select I.ID, I.NAME, I.DEPARTMENT, I.SALARY from
(select avg(BUDGET) as averageBudget from DEPARTMENT) as BUDGET, Instructor as I
where I.SALARY > BUDGET.averageBudget;
```

Output

InstructorID	Name	Department	Salary
44547	Smith	Computer Science	95000
48147	Erik	Mechanical	80000

Explanation: The average budget of all departments from the department relation is 70000. Erik and Smith are the only instructors in the instructor relation whose salary is more than 70000 and therefore are present in the output relation.

Last Updated : 11 Apr, 2022

40

Similar Reads

1. Difference between Having clause and Group by clause
2. SQL | Distinct Clause
3. SQL | WHERE Clause
4. Top Clause in Microsoft SQL Server
5. SQL | Union Clause
6. SQL | WITH clause
7. SQL | Except Clause
8. SQL | OFFSET-FETCH Clause
9. SQL | LIMIT Clause
10. SQL | Intersect & Except clause

Previous Next

Article Contributed By :

Mayank Kumar 12
Mayank Kumar 12

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : Dharmesh Singh 2, 19eucs091

Nested Queries in SQL

Prerequisites : Basics of SQL

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use **STUDENT, COURSE, STUDENT_COURSE** tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

  IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_ID**s for finding **S_ID**s from **STUDENT_COURSE** TABLE.

  STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'

Select **C_ID** from **COURSE** where **C_NAME** = ‘DSA’ or **C_NAME** = ‘DBMS’

STEP 2: Using **C_ID** of step 1 for finding **S_ID**

Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME** = ‘DSA’ or **C_NAME**=’DBMS’);

The inner query will return a set with members C1 and C3 and outer query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of **STUDENT**s who have either enrolled in ‘DSA’ or ‘DBMS’, it can be done as:

Select S_NAME from **STUDENT** where **S_ID** IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**=’DSA’ or **C_NAME**=’DBMS’));

NOT IN: If we want to find out **S_ID**s of **STUDENT**s who have neither enrolled in ‘DSA’ nor in ‘DBMS’, it can be done as:

Select **S_ID** from **STUDENT** where **S_ID** NOT IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**=’DSA’ or **C_NAME**=’DBMS’));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S_ID**s where **S_ID** is not a member of set (S1, S2 and S4). So it will return S3.

- **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out **S_NAME** of **STUDENT**s who are enrolled in **C_ID** ‘C1’, it can be done with the help of co-related nested query as:
  Select S_NAME from **STUDENT** S where EXISTS

  (select * from **STUDENT_COURSE** SC where S.**S_ID**=SC.**S_ID** and SC.**C_ID**=’C1’);

For each row of **STUDENT** S, it will find the rows from **STUDENT_COURSE** where S.**S_ID** = SC.**S_ID** and SC.**C_ID**='C1'. If for a **S_ID** from **STUDENT** S, atleast a row exists in **STUDENT_COURSE** SC with **C_ID**='C1', then inner query will return true and corresponding **S_ID** will be returned as output.

This article has been contributed by Sonal Tuteja.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 28 Jun, 2021

109

Similar Reads

1. Configure SQL Jobs in SQL Server using T-SQL
2. SQL vs NO SQL vs NEW SQL
3. Decision Making in PL/SQL (if-then , if-then-else, Nested if-then, if-then-elsif-then-else)
4. SQL queries on clustered and non-clustered Indexes
5. SQL queries on FILM Database
6. Production databases in SQL queries
7. SQL Concepts and Queries
8. How to Compare Two Queries in SQL
9. SQL | Difference between functions and stored procedures in PL/SQL
10. Difference between T-SQL and PL-SQL

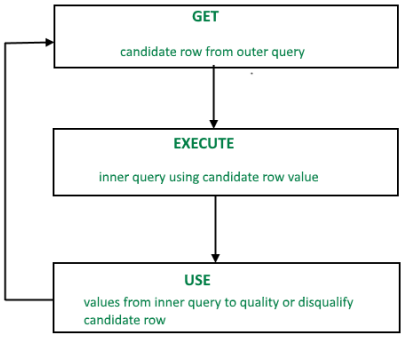
Previous Next

Article Contributed By :

SQL Correlated Subqueries

MrinalVerma

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

```
SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
                    (SELECT column1, column2
                     FROM table2
                     WHERE expr1 =
                               outer.expr2);
```

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

Nested Subqueries Versus Correlated Subqueries :

With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

> ***NOTE:*** *You can also use the* ***ANY*** *and* ***ALL*** *operator in a correlated subquery.* ***EXAMPLE of Correlated Subqueries :*** *Find all the employees who earn more than the average salary in their department.*


```
SELECT last_name, salary, department_id
 FROM employees outer
 WHERE salary >
                (SELECT AVG(salary)
                 FROM employees
                 WHERE department_id =
                        outer.department_id group by department_id);
```

Other use of correlation is in **UPDATE** and **DELETE**

CORRELATED UPDATE :

```
UPDATE table1 alias1
 SET column = (SELECT expression
               FROM table2 alias2
               WHERE alias1.column =
                     alias2.column);
```

Use a correlated subquery to update rows in one table based on rows from another table.

CORRELATED DELETE :

```
DELETE FROM table1 alias1
 WHERE column1 operator
               (SELECT expression
```

```
FROM table2 alias2
WHERE alias1.column = alias2.column);
```

Use a correlated subquery to delete rows in one table based on the rows from another table.

Using the EXISTS Operator :

The EXISTS operator tests for existence of rows in the results set of the subquery. If a subquery row value is found the condition is flagged **TRUE** and the search does not continue in the inner query, and if it is not found then the condition is flagged **FALSE** and the search continues in the inner query.

EXAMPLE of using EXIST operator :

Find employees who have at least one person reporting to them.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS ( SELECT 'X'
FROM employees
WHERE manager_id =
outer.employee_id);
```

OUTPUT :

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
108	Greenberg	FI_MGR	100
114	Raphaely	PU_MAN	30
120	Weiss	ST_MAN	50
121	Fripp	ST_MAN	50
122	Kaufling	ST_MAN	50
123	Vollman	ST_MAN	50
More than 10 rows available. Increase rows selector to view more rows.			

10 rows returned in 0.05 seconds CSV Export

EXAMPLE of using NOT EXIST operator :

Find all departments that do not have any employees.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
FROM employees
```

```
WHERE department_id
= d.department_id);
```

OUTPUT :

DEPARTMENT_ID	DEPARTMENT_NAME
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
180	Construction
190	Contracting
200	Operations
210	IT Support
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.18 seconds CSV Export

Last Updated : 11 Dec, 2022

39

Similar Reads

1. Difference between Nested Subquery, Correlated Subquery and Join Operation
2. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
3. Configure SQL Jobs in SQL Server using T-SQL
4. SQL | Procedures in PL/SQL
5. SQL | Difference between functions and stored procedures in PL/SQL
6. SQL SERVER – Input and Output Parameter For Dynamic SQL
7. Difference between SQL and T-SQL
8. SQL Server | Convert tables in T-SQL into XML
9. SQL SERVER | Bulk insert data from csv file using T-SQL command
10. SQL - SELECT from Multiple Tables with MS SQL Server

Previous Next

SQL Join vs Subquery

harikamoluguram27

What are Joins?

A join is a query that combines records from two or more tables. A join will be performed whenever multiple tables appear in the FROM clause of the query. The select list of the query can select any columns from any of these tables. If join condition is omitted or invalid then a Cartesian product is formed. If any two of these tables have a column name in common, then must qualify these columns throughout the query with table or table alias names to avoid ambiguity. Most join queries contain at least one join condition, either in the FROM clause or in the WHERE clause.

what is Subquery?

A Subquery or Inner query or Nested query is a query within SQL query and embedded within the WHERE clause. A Subquery is a SELECT statement that is embedded in a clause of another SQL statement. They can be very useful to select rows from a table with a condition that depends on the data in the same or another table. A Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. The subquery can be placed in the following SQL clauses they are WHERE clause, HAVING clause, FROM clause.

Advantages Of Joins:

- The advantage of a join includes that it executes faster.
- The retrieval time of the query using joins almost always will be faster than that of a subquery.

- By using joins, you can minimize the calculation burden on the database i.e., instead of multiple queries using one join query. This means you can make better use of the database's abilities to search through, filter, sort, etc.

Disadvantages Of Joins:

- Disadvantage of using joins includes that they are not as easy to read as subqueries.
- More joins in a query means the database server has to do more work, which means that it is more time consuming process to retrieve data
- As there are different types of joins, it can be confusing as to which join is the appropriate type of join to use to yield the correct desired result set.
- Joins cannot be avoided when retrieving data from a normalized database, but it is important that joins are performed correctly, as incorrect joins can result in serious performance degradation and inaccurate query results.

Advantages Of Subquery:

- Subqueries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand and code maintenance is also at ease.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions.

Disadvantages of Subquery:

- The optimizer is more mature for MYSQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as join.
- We cannot modify a table and select from the same table within a subquery in the same SQL statement.

Conclusion : A subquery is easier to write, but a joint might be better optimized by the server. For example a Left Outer join typically works faster because servers optimize it.

Last Updated : 07 Nov, 2022

17

Similar Reads

1. Difference between Nested Subquery, Correlated Subquery and Join Operation
2. SQL | Join (Cartesian Join & Self Join)

Difference between Nested Subquery, Correlated Subquery and Join Operation

deekshajaindodya

Join Operation :
Join operation is a binary operation used to combine data or rows from two or more tables based on a common field between them. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.

Example –

```
Orders (OrderID, CustomerID, OrderDate);
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * from Customers JOIN Orders
ON Orders.CustomerID=Customers.CustomerID;
```

Subquery
When a query is included inside another query, the Outer query is known as Main Query, and Inner query is known as Subquery.

- **Nested Query –**
  In Nested Query, Inner query runs first, and only once. Outer query is executed with result from Inner query.Hence, Inner query is used in execution of Outer query.
  Example –

```
Orders (OrderID, CustomerID, OrderDate);
```

```
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * FROM Customers WHERE
CustomerID IN (SELECT CustomerID FROM Orders);
```

- **Correlated Query** –
  In Correlated Query, Outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query.
  Example –

```
Orders (OrderID, CustomerID, OrderDate);
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * FROM Customers where
EXISTS (SELECT CustomerID FROM Orders
WHERE Orders.CustomerID=Customers.CustomerID);
```

Application of Join Operation and Subquery :
To understand the difference between Nested Subquery, Correlated Subquery and Join Operation firstly we have to understand where we use subqueries and where to use joins.

- When we want to get data from multiple tables we use join operation.
  Example: Let's consider two relations as:

```
Employee (eId, eName, eSalary, dId);
Department (dId, dName, dLocation);
```

Now, we have to find employee names and Department name working at London Location. Here, we have to display eName from employee table and dName from Department table. Hence we have to use Join Operation.

```
SELECT e.eName, d.dName from Employee e,
Department d
where e.dId=d.dId and d.dLocation="London";
```

- When we want to get data from one table and condition is based on another table we can either use Join or Subquery. Now, we have to find employee names working at London Location.
  Here, we have to display only eName from employee table hence we can use either Join Operation or Subquery
  Using Join Operation –

```
SELECT e.eName from Employee e, Department d
where e.dId=d.dId and d.dLocation="London";
```

Using Subquery –

```
SELECT eName from Employee
where dId=(SELECT dId from Department where dLocation="London");
```

After understanding the basic difference between Join and Subqueries, Now we will understand the difference between Nested Subquery, Correlated Subquery and Join Operation.

Difference between Nested Query, Correlated Query and Join Operation :

Parameters	Nested Query	Correlated Query	Join Operation
Definition	In Nested query, a query is written inside another query and the result of inner query is used in execution of outer query.	In Correlated query, a query is nested inside another query and inner query uses values from outer query.	Join operation is used to combine data or rows from two or more tables based on a common field between them.INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.
Approach	Bottom up approach i.e. Inner query runs first, and only once. Outer query is executed with result from Inner query.	Top to Down Approach i.e. Outer query executes first and for every Outer query row Inner query is executed.	It is basically cross product satisfying a condition.
Dependency	Inner query execution is not dependent on Outer query.	Inner query is dependent on Outer query.	There is no Inner Query or Outer Query. Hence, no dependency is there.
Performance	Performs better than Correlated	Performs slower than both Nested	By using joins we maximize the calculation

Parameters	Nested Query	Correlated Query	Join Operation
	Query but is slower than Join Operation.	Query and Join operations as for every outer query inner query is executed.	burden on the database but joins are better optimized by the server so the retrieval time of the query using joins will almost always be faster than that of a subquery.

Last Updated : 28 Dec, 2020

26

Similar Reads

1. Difference between Nested Loop Join and Hash Join
2. Difference between Nested Loop join and Sort Merge Join
3. SQL | Join (Cartesian Join & Self Join)
4. SQL Join vs Subquery
5. Join operation Vs Nested query in DBMS
6. Difference between Inner Join and Outer Join in SQL
7. Difference between Natural join and Inner Join in SQL
8. Difference between Natural join and Cross join in SQL
9. Difference between Hash Join and Sort Merge Join
10. Difference between “INNER JOIN” and “OUTER JOIN”

Previous Next

Article Contributed By :

deekshajaindodya
deekshajaindodya